Моделювання складних систем у Python

Соловйов В.М

Бєлінський А.О.

2023-09-26

Зміст

# Передмова

Це книга з моделювання складних систем засобами мови програмування Python.

# 1. Вступ

Задекларований перехід економіки України на інноваційний шлях розвитку в умовах обмеженості ресурсів, у першу чергу фінансових, вимагає зосередження їх на найбільш перспективних напрямках, де імовірність досягнення конкурентного успіху є найбільшою. Як свідчить практика, такого роду оцінки доцільно виконувати із застосуванням економіко-математичних моделей. Множинність шляхів вибору моделей і методів актуалізує необхідність розуміння основних тенденцій розвитку наукового знання і ключових досягнень.

Говорячи мовою синергетики, необхідно виділити такі параметри порядку — провідні змінні, які з плином часу починають визначати динаміку і розвиток складної системи, і підпорядковувати собі її інші параметри, які призведуть до ефективного спрощення складного об’єкта.

Все більш очевидною є революція, що почалася в природничих та гуманітарних науках і пов’язана з вивченням феномена самоорганізації і дослідженням мережних структур. Мережна парадигма складності обумовлена великим значенням, яке придбали такі об’єкти, і тим, що на початку ХХІ століття очевидною стала разюча аналогія в топології мережних структур, що виникають завдяки активному використанню інформаційно-телекомунікаційних, гуманітарних, управлінських, військових технологій.

Мережі стали одним з двигунів економіки. У своїй історії людство долало різні бар’єри, створюючи нові матеріали, технології, об’єкти. Однак “мережна економіка” зіткнулася з обмеженнями самої людини — так званим “когнітивним бар’єром”. Як показали психологи, людина в змозі активно, творчо взаємодіяти з 5-7 людьми (з рештою опосередковано або стандартно, незалежно від того, скільки у нього друзів у соціальній мережі). Вона може одночасно стежити за 5-7 змінними (незалежно від того, наскільки великий обсяг інформації їй доступний). Приймаючи рішення, вона може зважити 5-7 факторів (скільки б даних у неї не було).

Зліт нової економіки в США в 1990-х роках, пов’язаний багато в чому з інтернет- компаніями, породив ілюзії, що капіталізація мережних структур пропорційна числу зв’язків між вузлами, тобто квадрату числа вузлів . Однак, коли на початку 2000-х років міхур «нової економіки» лопнув (криза «доткомів - .com»), то виявилося, що реальна капіталізація мала зовсім іншу залежність від числа об’єктів, пов’язаних мережею — . Інакше кажучи, не «всі зв’язуються з усіма», а майже всі взаємодіють з декількома дуже великими вузлами- хабами, які вже тісно пов’язуються між собою. Подібним же чином виявляється влаштована інфраструктура більшості складних систем, незалежно від їх природи.

Черговим викликом для нас стала нова індустріальна революція, актуалізована давоським форумом 2016 року. Згідно з опитуванням 800 лідерів технологічних компаній, проведеним спеціально для форуму в Давосі, ключовими драйверами змін стануть хмарні технології, розвиток способів збору і аналізу Big Data, краудсорсінг, шерінгова економіка і біотехнології.

Очевидно, що революційні вимоги Індустрії 4.0 потребують нових парадигм моделювання соціально-економічних систем. На наш погляд, такою парадигмою може стати мережна парадигма складних систем.

Мережні технології змінили обчислювальну математику, системний аналіз, інформаційні технології. В останні роки було реалізовано кілька грандіозних мережних проектів, в яких поставлена задача вирішувалася завдяки спільним діям сотень тисяч або навіть мільйонів комп’ютерів. Це і криптографічні проблеми, і пошук ліків проти раку, заснований на математичному моделюванні взаємодії різних речовин з клітинами. Це розподілений аналіз даних космічних експериментів та обробка результатів, отриманих на Великому адронному колайдері.

У зв’язку з новою мережною парадигмою складності перед фахівцями з моделювання виникають принципово нові, актуальні задачі. Ось, на наш погляд, тільки деякі з них:

* дослідження надійності, робастності кіберфізичних мереж відносно випадкових помилок та направлених атак;
* аналіз когнітивних можливостей складних систем;
* моделювання мультиплексних мереж;
* вплив нанотехнологій на формування наноекономіки та ін.

Сьогодні вже зрозуміло, що відповідних інновацій вимагає і система освіти. На початку комп’ютерної ери основну цінність і проблеми становили власне комп’ютери (hardware) і акцент робився на підготовку фахівців з обчислювальної техніки. Потім величезне значення набуло програмне забезпечення (software) і була розпочата підготовка дослідників у цій галузі (computer science) та інженерів-програмістів (computer engineering). У даний час на перший план виходять фахівці з мережних технологій (NetWare). Саме таких фахівців треба починати готувати у провідних національних університетах. За оцінками експертів з Індустрії 4.0 основними трендами освітньої сфери на двадцятирічному горизонті стануть: поширення цінностей мережної культури, прагматизація освіти, автоматизація рутинних інтелектуальних операцій, розвиток індустрії поліпшення когнітивних здібностей, боротьба за таланти, зростання значущості глобальних людських цінностей, уваги до природи і дбайливого поводження з ресурсами.

**Мета курсу** моделювання складних систем — формування системи теоретичних знань та практичних навичок щодо моделювання структурних і динамічних властивостей систем різної природи як засобу дослідження та управління складними явищами на макро-, мезо- й мікрорівнях. Протягом останніх десяти-п’ятнадцяти років відбулися відчутні зміни в розумінні фундаментальних закономірностей складних систем. Виявилось, що такі складні системи мають універсальні емерджентні властивості, які не знаходять адекватного розуміння у рамках традиційних парадигм. Тому для аналізу все активніше використовуються сучасні методи та моделі фундаментальних наук, які у поєднанні з новітніми досягненнями в галузі інформаційних технологій та досить ємними базами даних (мільйони записів навіть в базах некомерційного призначення) забезпечили значний прогрес у розумінні та квантифікації природи цих систем. Сюди відносяться методи фрактального і мультифрактального аналізу, дослідження рекурентних властивостей динамічних систем, нелінійної динаміки, теорії хаосу і біфуркацій тощо. З’явились нові «кількісні» напрямки економіки: математична економіка, фізична економіка, еконофізика та ін. Деякі з таких моделей, не знайшовши поки що відображення у навчальній літературі, включені до цієї роботи.

**Наукову основу курсу** складають теоретичні моделі, математичний апарат, сучасні концепції та парадигми, які визначають підходи до вивчення характеристик складних систем. Курс базується на знаннях, одержаних при вивченні дисциплін математичного циклу, основ теорії систем та системного аналізу, моделювання, фінансового аналізу, макро- і мікроекономіки.

**Завдання курсу** — оволодіння теоретичними знаннями та інструментарієм моделювання складних систем, вивчення підходів до дослідження й аналізу, методів прогнозування їхнього розвитку, управління розвитком та функціонуванням складних систем у різних умовах функціонування.

**Предметом курсу** є математичні моделі і методи дослідження складних систем різної природи.

У результаті вивчення дисципліни студент повинен:

* знати структурні та динамічні характеристики складної системи; моделі прогнозування характеристик системи; основні методи оцінки якості функціонування; методи оцінки структурних змін; методи дослідження та моделювання складних природних та штучних систем;
* вміти здійснювати класифікацію характеристик складної системи, проводити порівняльний аналіз методів прогнозування; оцінити якість функціонування ієрархічної системи; визначити катастрофічні зміни в системі, які описуються рівняннями динаміки, визначити джерела структурних катастроф;
* дослідити та проаналізувати комплекс моделей складної системи;
* бути ознайомленим з сучасними напрямками розвитку сучасних теорій та парадигм, які використовуються для дослідження якісних характеристик динамічних систем.

Система контролю якості навчання студентів містить такі заходи:

* модульний контроль;
* проведення контрольних робіт;
* контроль теоретичних знань у ході практичних занять;
* виконання індивідуальних завдань на лабораторних заняттях.

Організація самостійної роботи студентів передбачає підготовку до семінарських занять, проведення індивідуальних консультацій, виконання курсової роботи.

Відсутність альтернативних підручників і методичних посібників з курсу моделювання складних систем спонукала авторів розробити методичні вказівки до виконання практичних робіт з даного курсу. Вони спираються на деяке авторське бачення, певні висновки мають дискусійний характер. Автори будуть вдячні за бачення, спрямоване на покращення змісту та методики викладання дисципліни.

# 2. Лабораторна робота № 1

**Тема.** Аналіз флуктуацій часового ряду.

**Мета.** Навчитися використовувати аналіз флуктуацій та його похідні для отримання нелінійних характеристик часового ряду.

## 2.1 Теоретичні відомості

### 2.1.1 Аналіз динаміки прибутків, модулів прибутків та волатильностей

Останнім часом вчені все більше цікавляться економічними часовими рядами, і відбувається це за кількох причин, зокрема: (1) економічні часові ряди, такі як індекси акцій, курсів валют, залежать від розвитку великої кількості взаємодіючих систем, і є прикладами складних систем, що широко вивчаються у науці; (2) з’явилась велика кількість доступних баз з даними про економічні системи, що містять інформацію з різними часовими шкалами (починаючи з 1 хвилини і закінчуючи 1 роком). Внаслідок цього вже на даний час існує також велика кількість розроблених методів (зокрема, у статистичній фізиці), спрямованих на отримання характеристик цін акцій чи курсів валют, що еволюціонують у часі.

Дослідження, проведені над часовими рядами, показують, що стохастичний процес, який лежить у основі зміни ціни, характеризується кількома ознаками. Розподіл зміни ціни має виділений хвіст порівняно із Гаусовим розподілом. Функція автокореляції зміни ціни спадає експоненційно з певним характерним часом. Однак, виявляється, що амплітуда зміни ціни, виміряна за абсолютними значеннями чи квадратами цін, показує степеневі кореляції з довго часовою персистентністю аж до кількох місяців, або навіть років. Такі довгочасові залежності краще моделюються з використанням «додаткового процесу», що в економічній літературі часто називається волатильністю. Волатильність змін ціни акції є мірою того, як сильно ринок схильний до флуктуацій, тобто відхилень ціни від попередніх значень.

Першим кроком при проведенні аналізу є побудова оцінювача волатильності. Ми будемо отримувати волатильність як локальне середнє модуля зміни ціни.

Розуміння статистичних властивостей волатильності має також важливе практичне застосування. Волатильність є інтересом торговців, оскільки визначає ризик і є ключовим входом практично до всіх моделей цін опціонів (вторинного цінного паперу), включаючи і класичну модель Блека-Шоулза. Без задовільних методів оцінювання волатильності трейдерам було б надзвичайно важко визначати ситуації, в яких опціони попадають в недооцінку чи переоцінку.

### 2.1.2 Визначення волатильності

Термін волатильність представляє узагальнену міру величини ринкових флуктуацій (відхилень). У літературі існує досить багато визначень волатильності, проте ми будемо використовувати наступне: *волатильність є локальним середнім модуля зміни ціни на відповідному часовому інтервалі , що є рухомим параметром нашої оцінки*. Для індексу визначимо зміну ціни як зміну логарифмів індексів,

де є часовим інтервалом затримки. Величину (1) називають прибутковістю (return). Якщо використовувати границі, то малі зміни приблизно відповідають змінам, визначеним другою рівністю. Ми лише підраховуємо час роботи ринку, викидаємо ночі, вихідні та свята із набору даних, тобто, вважається, що ринок працює без перерв.

Модуль описує амплітуду флуктуацій. У порівнянні із значеннями їх модуль не показує глобальних трендів, але великі значення відповідають крахам та великим миттєвим змінам на ринках.

Визначимо волатильність як середнє від для часових вікон , тобто

де є цілим числом. Таке визначення може бути ще узагальнене заміною на , де дає більш виражені великі значення , в той час як виділяє малі значення .

У цьому визначенні волатильності використовується два параметри, та . Параметр є шаблонним (чи модельним) часовим інтервалом для даних, а параметр є кроком переміщення часового вікна. Зауважимо, що вказане визначення волатильності має внутрішню помилку, а саме: вибір більшого часового інтервалу веде до збільшення точності визначення волатильності. Однак, велике значення також включає погане розбиття часу на інтервали, що веде, у свою чергу, до врахування не всієї прихованої у ряді інформації.

### 2.1.3 Визначення кореляцій

Для визначення кореляцій часового ряду використовується функція автокореляції. Саме поняття *автокореляції* означає кореляцію часового ряду самого з собою (між попередніми та наступними значеннями). Автокореляцію іноді називають *послідовною кореляцією*, що означає кореляцію між членами ряду чисел, розташованих у певному порядку. Також синонімами цього терміну є *лагова кореляція* та *персистентність*. Наприклад, часто зустрічається автокореляція геофізичних процесів, що означає перенесення залишкового процесу на наступні часові проміжки.

Позитивно автокорельований часовий ряд часто називають персистентним, що значить існування тенденції слідування великих значень за великими та малих за малими, інакше позитивно корельований часовий ряд можна назвати інертним.

Візьмемо пар спостережень двох змінних та . Коефіцієнт кореляції між парами та визначається як

де сума знаходиться по всім спостереженням.

Таким же чином можна визначати й автокореляцію, або ж кореляцію всередині досліджуваного часового ряду. Для автокореляції першого порядку береться лаг (часова затримка), рівний 1 часовій одиниці. Таким чином, автокореляція першого порядку використовує перші спостережень , та наступні спостережень .

Кореляція між та визначається наступним чином:

де — це середнє для досліджуваного періоду.

Рівняння (4) може бути узагальнене для отримання кореляції між спостереженнями, розділеними часовими інтервалами:

Значення називається коефіцієнтом автокореляції з лагом . Графік функції автокореляції як залежності від також називають корелограмою.

## 2.2 Хід роботи

Для подальшої роботи з моделювання складних систем візьмемо з основу бібліотеку yfinance, що дозволяє працювати з даними фінансових ринків засобами мови програмування Python.

|  |
| --- |
| Примітка |
| **Yahoo!, Y!Finance, and Yahoo! finance є зареєстрованими товарними знаками Yahoo, Inc.**  yfinance не є афілійованим, схваленим або перевіреним Yahoo, Inc. Це інструмент з відкритим вихідним кодом, який використовує загальнодоступні API Yahoo, і призначений для дослідницьких та освітніх цілей.  Ви повинні звернутися до умов використання Yahoo! ([сюди](https://policies.yahoo.com/us/en/yahoo/terms/product-atos/apiforydn/index.htm), [сюди](https://legal.yahoo.com/us/en/yahoo/terms/otos/index.html) і [сюди](https://policies.yahoo.com/us/en/yahoo/terms/index.htm)) для отримання детальної інформації про ваші права на використання фактично завантажених даних. Пам’ятайте — фінансовий API Yahoo! призначений лише для особистого використання. |

Для встановлення бібліотеки yfinance можете скористатися наступною командою:

!pip install yfinance --upgrade --no-cache-dir

[Гітхаб репозиторій](https://github.com/ranaroussi/yfinance) містить більше інформації по самій бібліотеці та помилкам, що можуть виникнути та їх потенційним рішенням.

### 2.2.1 Вступ до модуля Ticker()

Перш за все імпортуємо бібліотеку yfinance за допомогою наступної команди:

import yfinance as yf

Модуль Ticker() дозволяє отримувати ринкові та метадані для цінного паперу, використовуючи Python:

msft = yf.Ticker("MSFT")  
print(msft)

yfinance.Ticker object <MSFT>

Можна вилучити всю інформацію по досліджуваному індексу:

# отримати інформацію по індексу  
print(msft.info)

{'address1': 'One Microsoft Way', 'city': 'Redmond', 'state': 'WA', 'zip': '98052-6399', 'country': 'United States', 'phone': '425 882 8080', 'website': 'https://www.microsoft.com', 'industry': 'Software—Infrastructure', 'industryDisp': 'Software—Infrastructure', 'sector': 'Technology', 'sectorDisp': 'Technology', 'longBusinessSummary': 'Microsoft Corporation develops and supports software, services, devices and solutions worldwide. The Productivity and Business Processes segment offers office, exchange, SharePoint, Microsoft Teams, office 365 Security and Compliance, Microsoft viva, and Microsoft 365 copilot; and office consumer services, such as Microsoft 365 consumer subscriptions, Office licensed on-premises, and other office services. This segment also provides LinkedIn; and dynamics business solutions, including Dynamics 365, a set of intelligent, cloud-based applications across ERP, CRM, power apps, and power automate; and on-premises ERP and CRM applications. The Intelligent Cloud segment provides server products and cloud services, such as azure and other cloud services; SQL and windows server, visual studio, system center, and related client access licenses, as well as nuance and GitHub; and enterprise services including enterprise support services, industry solutions, and nuance professional services. The More Personal Computing segment offers Windows, including windows OEM licensing and other non-volume licensing of the Windows operating system; Windows commercial comprising volume licensing of the Windows operating system, windows cloud services, and other Windows commercial offerings; patent licensing; and windows Internet of Things; and devices, such as surface, HoloLens, and PC accessories. Additionally, this segment provides gaming, which includes Xbox hardware and content, and first- and third-party content; Xbox game pass and other subscriptions, cloud gaming, advertising, third-party disc royalties, and other cloud services; and search and news advertising, which includes Bing, Microsoft News and Edge, and third-party affiliates. The company sells its products through OEMs, distributors, and resellers; and directly through digital marketplaces, online, and retail stores. The company was founded in 1975 and is headquartered in Redmond, Washington.', 'fullTimeEmployees': 221000, 'companyOfficers': [{'maxAge': 1, 'name': 'Mr. Satya Nadella', 'age': 55, 'title': 'Chairman & CEO', 'yearBorn': 1967, 'fiscalYear': 2022, 'totalPay': 12676750, 'exercisedValue': 0, 'unexercisedValue': 0}, {'maxAge': 1, 'name': 'Mr. Bradford L. Smith LCA', 'age': 63, 'title': 'Pres & Vice Chairman', 'yearBorn': 1959, 'fiscalYear': 2022, 'totalPay': 4655274, 'exercisedValue': 0, 'unexercisedValue': 0}, {'maxAge': 1, 'name': 'Ms. Amy E. Hood', 'age': 50, 'title': 'Exec. VP & CFO', 'yearBorn': 1972, 'fiscalYear': 2022, 'totalPay': 4637915, 'exercisedValue': 0, 'unexercisedValue': 0}, {'maxAge': 1, 'name': 'Mr. Judson Althoff', 'age': 49, 'title': 'Exec. VP & Chief Commercial Officer', 'yearBorn': 1973, 'fiscalYear': 2022, 'totalPay': 4428268, 'exercisedValue': 0, 'unexercisedValue': 0}, {'maxAge': 1, 'name': 'Mr. Christopher David Young', 'age': 50, 'title': 'Exec. VP of Bus. Devel., Strategy & Ventures', 'yearBorn': 1972, 'fiscalYear': 2022, 'totalPay': 4588876, 'exercisedValue': 0, 'unexercisedValue': 0}, {'maxAge': 1, 'name': 'Ms. Alice L. Jolla', 'age': 55, 'title': 'Corp. VP & Chief Accounting Officer', 'yearBorn': 1967, 'exercisedValue': 0, 'unexercisedValue': 0}, {'maxAge': 1, 'name': 'Mr. James Kevin Scott', 'age': 50, 'title': 'Exec. VP of AI & CTO', 'yearBorn': 1972, 'exercisedValue': 0, 'unexercisedValue': 0}, {'maxAge': 1, 'name': 'Brett Iversen', 'title': 'VP of Investor Relations', 'exercisedValue': 0, 'unexercisedValue': 0}, {'maxAge': 1, 'name': 'Mr. Hossein Nowbar', 'title': 'Chief Legal Officer', 'exercisedValue': 0, 'unexercisedValue': 0}, {'maxAge': 1, 'name': 'Mr. Frank X. Shaw', 'title': 'Chief Communications Officer', 'exercisedValue': 0, 'unexercisedValue': 0}], 'auditRisk': 6, 'boardRisk': 3, 'compensationRisk': 3, 'shareHolderRightsRisk': 2, 'overallRisk': 2, 'governanceEpochDate': 1693526400, 'compensationAsOfEpochDate': 1672444800, 'maxAge': 86400, 'priceHint': 2, 'previousClose': 328.65, 'open': 329.51, 'dayLow': 325.03, 'dayHigh': 329.59, 'regularMarketPreviousClose': 328.65, 'regularMarketOpen': 329.51, 'regularMarketDayLow': 325.03, 'regularMarketDayHigh': 329.59, 'dividendRate': 2.72, 'dividendYield': 0.0083, 'exDividendDate': 1692144000, 'payoutRatio': 0.2748, 'fiveYearAvgDividendYield': 1.05, 'beta': 0.904564, 'trailingPE': 33.579372, 'forwardPE': 25.809818, 'volume': 8304959, 'regularMarketVolume': 8304959, 'averageVolume': 24743406, 'averageVolume10days': 19741410, 'averageDailyVolume10Day': 19741410, 'bid': 325.98, 'ask': 326.04, 'bidSize': 1300, 'askSize': 1100, 'marketCap': 2420020871168, 'fiftyTwoWeekLow': 213.43, 'fiftyTwoWeekHigh': 366.78, 'priceToSalesTrailing12Months': 11.419771, 'fiftyDayAverage': 332.0454, 'twoHundredDayAverage': 293.46735, 'trailingAnnualDividendRate': 2.72, 'trailingAnnualDividendYield': 0.008276282, 'currency': 'USD', 'enterpriseValue': 2409976823808, 'profitMargins': 0.34146, 'floatShares': 7423522720, 'sharesOutstanding': 7429760000, 'sharesShort': 38317589, 'sharesShortPriorMonth': 35852374, 'sharesShortPreviousMonthDate': 1690761600, 'dateShortInterest': 1693440000, 'sharesPercentSharesOut': 0.0052, 'heldPercentInsiders': 0.00052, 'heldPercentInstitutions': 0.73212, 'shortRatio': 1.82, 'shortPercentOfFloat': 0.0052, 'impliedSharesOutstanding': 7429760000, 'bookValue': 27.748, 'priceToBook': 11.738501, 'lastFiscalYearEnd': 1688083200, 'nextFiscalYearEnd': 1719705600, 'mostRecentQuarter': 1688083200, 'earningsQuarterlyGrowth': 0.2, 'netIncomeToCommon': 72361000960, 'trailingEps': 9.7, 'forwardEps': 12.62, 'pegRatio': 2.08, 'lastSplitFactor': '2:1', 'lastSplitDate': 1045526400, 'enterpriseToRevenue': 11.372, 'enterpriseToEbitda': 23.622, '52WeekChange': 0.37539232, 'SandP52WeekChange': 0.17256784, 'lastDividendValue': 0.68, 'lastDividendDate': 1692144000, 'exchange': 'NMS', 'quoteType': 'EQUITY', 'symbol': 'MSFT', 'underlyingSymbol': 'MSFT', 'shortName': 'Microsoft Corporation', 'longName': 'Microsoft Corporation', 'firstTradeDateEpochUtc': 511108200, 'timeZoneFullName': 'America/New\_York', 'timeZoneShortName': 'EDT', 'uuid': 'b004b3ec-de24-385e-b2c1-923f10d3fb62', 'messageBoardId': 'finmb\_21835', 'gmtOffSetMilliseconds': -14400000, 'currentPrice': 325.7199, 'targetHighPrice': 440.0, 'targetLowPrice': 232.0, 'targetMeanPrice': 386.95, 'targetMedianPrice': 400.0, 'recommendationMean': 1.8, 'recommendationKey': 'buy', 'numberOfAnalystOpinions': 43, 'totalCash': 111256002560, 'totalCashPerShare': 14.974, 'ebitda': 102022995968, 'totalDebt': 79441002496, 'quickRatio': 1.536, 'currentRatio': 1.769, 'totalRevenue': 211914997760, 'debtToEquity': 38.522, 'revenuePerShare': 28.46, 'returnOnAssets': 0.14245, 'returnOnEquity': 0.38824, 'freeCashflow': 47268999168, 'operatingCashflow': 87581999104, 'earningsGrowth': 0.202, 'revenueGrowth': 0.083, 'grossMargins': 0.6892, 'ebitdaMargins': 0.48143002, 'operatingMargins': 0.41772997, 'financialCurrency': 'USD', 'trailingPegRatio': 2.4062}

Можна вилучити ринкові значення за максимальний період часу:

# отримати ринкові історичні значення індексу  
msft.history(period="max")

|  | Open | High | Low | Close | Volume | Dividends | Stock Splits |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Date |  |  |  |  |  |  |  |
| 1986-03-13 00:00:00-05:00 | 0.055004 | 0.063093 | 0.055004 | 0.060396 | 1031788800 | 0.0 | 0.0 |
| 1986-03-14 00:00:00-05:00 | 0.060396 | 0.063632 | 0.060396 | 0.062553 | 308160000 | 0.0 | 0.0 |
| 1986-03-17 00:00:00-05:00 | 0.062553 | 0.064172 | 0.062553 | 0.063632 | 133171200 | 0.0 | 0.0 |
| 1986-03-18 00:00:00-05:00 | 0.063632 | 0.064172 | 0.061475 | 0.062014 | 67766400 | 0.0 | 0.0 |
| 1986-03-19 00:00:00-05:00 | 0.062014 | 0.062553 | 0.060396 | 0.060936 | 47894400 | 0.0 | 0.0 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 2023-09-14 00:00:00-04:00 | 339.149994 | 340.859985 | 336.570007 | 338.700012 | 20267000 | 0.0 | 0.0 |
| 2023-09-15 00:00:00-04:00 | 336.920013 | 337.399994 | 329.649994 | 330.220001 | 37666900 | 0.0 | 0.0 |
| 2023-09-18 00:00:00-04:00 | 327.799988 | 330.399994 | 326.359985 | 329.059998 | 16834200 | 0.0 | 0.0 |
| 2023-09-19 00:00:00-04:00 | 326.170013 | 329.390015 | 324.510010 | 328.649994 | 16505900 | 0.0 | 0.0 |
| 2023-09-20 00:00:00-04:00 | 329.510010 | 329.589996 | 325.029999 | 325.720001 | 8307027 | 0.0 | 0.0 |

Окрім цього, yfinance дозволяє отримати інформацію по дивідентам та сплітам фінансового індексу:

# показувати дії (дивіденди, спліти)  
msft.actions

|  | Dividends | Stock Splits |
| --- | --- | --- |
| Date |  |  |
| 1987-09-21 00:00:00-04:00 | 0.00 | 2.0 |
| 1990-04-16 00:00:00-04:00 | 0.00 | 2.0 |
| 1991-06-27 00:00:00-04:00 | 0.00 | 1.5 |
| 1992-06-15 00:00:00-04:00 | 0.00 | 1.5 |
| 1994-05-23 00:00:00-04:00 | 0.00 | 2.0 |
| ... | ... | ... |
| 2022-08-17 00:00:00-04:00 | 0.62 | 0.0 |
| 2022-11-16 00:00:00-05:00 | 0.68 | 0.0 |
| 2023-02-15 00:00:00-05:00 | 0.68 | 0.0 |
| 2023-05-17 00:00:00-04:00 | 0.68 | 0.0 |
| 2023-08-16 00:00:00-04:00 | 0.68 | 0.0 |

# продемонструвати дивіденти  
msft.dividends

Date  
2003-02-19 00:00:00-05:00 0.08  
2003-10-15 00:00:00-04:00 0.16  
2004-08-23 00:00:00-04:00 0.08  
2004-11-15 00:00:00-05:00 3.08  
2005-02-15 00:00:00-05:00 0.08  
 ...   
2022-08-17 00:00:00-04:00 0.62  
2022-11-16 00:00:00-05:00 0.68  
2023-02-15 00:00:00-05:00 0.68  
2023-05-17 00:00:00-04:00 0.68  
2023-08-16 00:00:00-04:00 0.68  
Name: Dividends, Length: 79, dtype: float64

# продемонструвати спліти  
msft.splits

Date  
1987-09-21 00:00:00-04:00 2.0  
1990-04-16 00:00:00-04:00 2.0  
1991-06-27 00:00:00-04:00 1.5  
1992-06-15 00:00:00-04:00 1.5  
1994-05-23 00:00:00-04:00 2.0  
1996-12-09 00:00:00-05:00 2.0  
1998-02-23 00:00:00-05:00 2.0  
1999-03-29 00:00:00-05:00 2.0  
2003-02-18 00:00:00-05:00 2.0  
Name: Stock Splits, dtype: float64

Для методу history() доступні наступні аргументи:

* **period**: період даних для завантаження (або використовуйте параметр period, або використовуйте start і end). Допустимі періоди: 1d, 5d, 1mo, 3mo, 6mo, 1y, 2y, 5y, 10y, ytd, max.
* **interval**: інтервал даних (внутрішньоденні дані не можуть перевищувати 60 днів) Допустимі інтервали 1m, 2m, 5m, 15m, 30m, 60m, 90m, 1h, 1d, 5d, 1wk, 1mo, 3mo.
* **start**: Якщо не використовується період — завантажте рядок дати початку у форматі (YYYY-MM-DD) або *datetime*.
* **end**: Якщо не використовується період — завантажте рядок дати закінчення (YYYY-MM-DD) або *datetime*.
* **prepost**: Включати в результати попередні та пост ринкові дані (За замовчуванням False).
* **auto\_adjust**: Автоматично налаштовувати всі OHLC (ціни відкриття, закриття, найбільшу та найменшу) (За замовчуванням True).
* **actions**: Завантажувати події дивідендів та дроблення акцій (За замовчуванням True).

### 2.2.2 Одночасне вивантаження декількох ринкових активів

Як і до цього, ви також можете завантажувати дані для кількох тикерів одночасно.

data = yf.download("SPY AAPL",   
 start="2017-01-01",   
 end="2017-04-30") # вивантажуємо дані,   
 # зберігаємо до змінної data  
  
data.head() # виводимо перші 5 рядків нашого масиву даних

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 2 of 2 completed

|  | Adj Close | | Close | | High | | Low | | Open | | Volume | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | AAPL | SPY | AAPL | SPY | AAPL | SPY | AAPL | SPY | AAPL | SPY | AAPL | SPY |
| Date |  |  |  |  |  |  |  |  |  |  |  |  |
| 2017-01-03 | 27.059307 | 200.629669 | 29.037500 | 225.240005 | 29.082500 | 225.830002 | 28.690001 | 223.880005 | 28.950001 | 225.039993 | 115127600 | 91366500 |
| 2017-01-04 | 27.029026 | 201.823288 | 29.004999 | 226.580002 | 29.127501 | 226.750000 | 28.937500 | 225.610001 | 28.962500 | 225.619995 | 84472400 | 78744400 |
| 2017-01-05 | 27.166475 | 201.662903 | 29.152500 | 226.399994 | 29.215000 | 226.580002 | 28.952499 | 225.479996 | 28.980000 | 226.270004 | 88774400 | 78379000 |
| 2017-01-06 | 27.469332 | 202.384430 | 29.477501 | 227.210007 | 29.540001 | 227.750000 | 29.117500 | 225.899994 | 29.195000 | 226.529999 | 127007600 | 71559900 |
| 2017-01-09 | 27.720936 | 201.716385 | 29.747499 | 226.460007 | 29.857500 | 227.070007 | 29.485001 | 226.419998 | 29.487499 | 226.910004 | 134247600 | 46939700 |

Для отримання конкретно цін закриття індексу **SPY**, вам варто використовувати наступну команду: data['Close']['SPY'] Але, якщо вам потребується згрупувати дані по їх символу, можна скористатися наступним записом:

data = yf.download("SPY AAPL",   
 start="2017-01-01",   
 end="2017-04-30",  
 group\_by="ticker")

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 2 of 2 completed

Тепер для звернення до цін закриття індексу **SPY**, вам треба використовувати наступний запис: data['SPY']['Close'].

Метод download() приймає додатковий параметр — threads для швидшої обробки великої кількості фінансових індексів одночасно.

Для подальшої роботи у даній лабораторній та подальших нас ще цікавитимуть наступні бібліотеки:

* **matplotlib**: комплексна бібліотека для створення статичних, анімованих та інтерактивних візуалізацій на Python. Matplotlib робить прості речі простими, а складні — можливими.
* **pandas**: програмна бібліотека, написана для мови програмування Python для маніпулювання та аналізу даних. Зокрема, вона пропонує структури даних та операції для маніпулювання числовими таблицями та часовими рядами.
* **numpy**: бібліотека, що додає підтримку великих багатовимірних масивів і матриць, а також велику колекцію високорівневих математичних функцій для роботи з цими масивами.
* **neurokit2**: зручна бібліотека, що забезпечує легкий доступ до розширених процедур обробки біосигналів. Дослідники та клініцисти без глибоких знань програмування або біомедичної обробки сигналів можуть аналізувати фізіологічні дані за допомогою лише двох рядків коду. Перевага даної бібліотеки полягає в тому, що вона надає функціонал, який можна використовувати не лише для біомедичних сигналів, але й для фінансових, фізичних тощо.

Встановити кожну з даних бібліотек можна в наступний спосіб: !pip install \*назва бібліотеки\*:

!pip install matplotlib pandas numpy neurokit2

Далі нам треба буде визначити стиль рисунків для виведення та збереження. Зробити це можна в наступний спосіб. Для використання подальшого стилю рисунків потребується встановити наступну бібліотеку:

# для встановлення останньої версії (із PyPI)  
!pip install SciencePlots

Імпортуємо кожну із зазначених бібліотек:

import matplotlib.pyplot as plt  
import pandas as pd   
import numpy as np  
import neurokit2 as nk  
import scienceplots  
  
# магічна команда для вбудування рисунків у середовище jupyter блокнотів  
%matplotlib inline

Виконаємо налаштування стилю наших подальших рисунків:

plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься  
 # для виведення рисунків  
  
params = {  
 'figure.figsize': (8, 6), # встановлюємо ширину та висоту рисунків   
 # за замовчуванням  
   
 'font.size': 22, # розмір фонтів рисунку  
 'lines.linewidth': 2, # товщина ліній  
 'axes.titlesize': 'small', # розмір титулки над рисунком  
 "font.family": "sans-serif", # сімейство стилів підписів   
 "font.serif": ["Times"], # стиль підпису  
 'savefig.dpi': 300 # якість збережених зображень  
}  
  
plt.rcParams.update(params) # оновлення стилю згідно налаштувань

Представлені налаштування є орієнтовними і можуть змінюватись у ході наступних лабораторних. Ви можете встановлювати власні налаштування. На сайті бібліотеки [matplotlib](https://matplotlib.org/stable/api/matplotlib_configuration_api.html) можна ознайомитись з усіма можливими командами.

Розглянемо можливість використання всіх згаданих показників у якості індикаторів або індикаторів-передвісників кризових явищ. Для прикладу завантажимо часовий ряд Біткоїна за період з 1 вересня 2015 по 1 березня 2020, використовуючи yfinance:

symbol = 'BTC-USD' # Символ індексу  
start = "2015-09-01" # Дата початку зчитування даних  
end = "2020-03-01" # Дата закінчення зчитування даних  
  
data = yf.download(symbol, start, end) # вивантажуємо дані  
time\_ser = data['Adj Close'].copy() # зберігаємо саме ціни закриття  
  
xlabel = 'time, days' # підпис по вісі Ох   
ylabel = symbol # підпис по вісі Оу

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

|  |
| --- |
| Важливо |
| Представлені в подальшому методи є універсальними. Іншими словами, ви можете їх використовувати не лише для фінансових часових рядів, але й для, біологічних, фізичних і для будь-яких інших систем, що існують в осяжній нами реальності та можуть бути репрезентовані у вигляду часового ряду. Може бути так, що наявні у вас дані, наприклад, представлені у форматі текстового документа (.txt). Нижче представлено приклад зчитування текстового файлу, що представляє залежність між напруженням і деформацією. Представлену далі залежність було отримано в результаті механічних випробувань певного металу. |

symbol = 'sMpa11' # Символ індексу  
  
path = "databases\sMpa11.txt" # шлях по якому здійснюється зчитування файлу  
data = pd.read\_csv(path, # зчитування даних   
 names=[symbol])  
time\_ser = data[symbol].copy() # копіюємо значення кривої   
 # "напруга-видовження" до окремої змінної  
  
xlabel = r'$\varepsilon$' # підпис по вісі Ох   
ylabel = symbol # підпис по вісі Оу

Тепер виведемо значення, що були зчитані з текстового документа:

fig, ax = plt.subplots() # Створюємо порожній графік  
ax.plot(time\_ser.index, time\_ser.values) # Додаємо дані до графіку  
ax.legend([symbol]) # Додаємо легенду  
ax.set\_xlabel(xlabel)  
ax.set\_ylabel(ylabel)  
  
plt.xticks(rotation=45) # оберт позначок по осі Ох на 45 градусів  
  
plt.savefig(f'{symbol}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 2.1: Діаграма деформації металевого матеріалу |

Для даного ряду, за потребою, можна виконувати подальші розрахунки.

|  |
| --- |
| Увага |
| Знову повертаємось до Біткоїна. Для відтворення подальших розрахунків з Біткоїном, блок коду в якому зчитувалась та виводилась крива “напруга-видовження” треба проігнорувати. |

Виведемо значення Біткоїна:

fig, ax = plt.subplots() # Створюємо порожній графік  
ax.plot(time\_ser.index, time\_ser.values) # Додаємо дані до графіку  
ax.legend([symbol]) # Додаємо легенду  
ax.set\_xlabel(xlabel)  
ax.set\_ylabel(ylabel)  
  
plt.xticks(rotation=45) # оберт позначок по осі Ох на 45 градусів  
  
plt.savefig(f'{symbol}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 2.2: Динаміка щоденних змін індексу Біткоїна |

Видно, що ряд нестаціонарний, що викликає певні ускладнення для подальшого аналізу. Тому перейдемо до прибутковостей, які вже є стаціонарними, а їх нормалізація стандартним відхиленням дозволяє легко порівнювати їх розподіл з розподілом Гауса.

Прибутковості розраховуватимуться згідно формулі (1). У Python ми використовуватимемо метод pct\_change() для знаходження прибутковостей, що доступний нам завдяки бібліотеці pandas.

Стандартизовані прибутковості визначаються наступним шляхом:

де відповідає середньому значенню прибутковостей за досліджуваний часовий інтервал, а представляє стандартне відхилення.

ret = time\_ser.copy() # копіюємо значення вихідного ряду для збереження   
 # його від змін  
  
ret = ret.pct\_change() # знаходимо прибутковості  
ret -= ret.mean() # вилучаємо середнє   
ret /= ret.std() # ділимо на стандартнє відхилення  
  
ret = ret.dropna().values # видаляємо всі можливі нульові значення

Виводимо отриманий результат:

fig, ax = plt.subplots() # Створюємо порожній графік  
ax.plot(time\_ser.index[1:], ret) # Додаємо дані до графіку  
ax.legend([symbol]) # Додаємо легенду  
ax.set\_xlabel(xlabel) # Додаємо підпис для вісі Ох  
ax.set\_ylabel(ylabel + ' прибутковості') # Додаємо підпис для вісі Оу  
ax.axhline(y = 3.0, color = 'r', linestyle = '--') # Додаємо горизонтальну лінію, що роз-  
 # межує 3 сигма події  
ax.axhline(y = -3.0, color = 'r', linestyle = '--') # Додаємо горизонтальну лінію, що роз-  
 # межує -3 сигма події  
  
plt.xticks(rotation=45) # оберт позначок по осі Ох на 45 градусів  
  
plt.savefig(f'Прибутковості{symbol}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 2.3: Нормалізовані прибутковості досліджуваного часового ряду |

Зверніть увагу, що флуктуації нормалізованих прибутковостей досить часто перевищують величину , що, як відомо, надзвичайно рідко спостерігається для незалежних подій. Цей факт можна відобразити шляхом порівняння функції розподілу нормалізованих флуктуацій з розподілом Гауса (рис. [Рисунок 2.4](#fig-btc-dist)). Очевидно, що хвости розподілу вихідного ряду містять значні флуктуації, вони досить помітні (часто кажуть “важкі” у порівнянні з самою “головою” розподілу).

Для побудови нормального розподілу скористаємось бібліотекою scipy. Встановити її можна по аналогії з попередніми бібліотеками.

# Для встановлення останньої версії scipy  
!pip install scipy

from scipy.stats import norm # імпорт модуля norm для побудови Гаусового розподілу

Функція щільності ймовірності для norm має наступний вид:

для дійсних значень .

mu, sigma = norm.fit(ret)  
  
x = np.linspace(ret.min(), ret.max(), 10000) # Генеруємо 10000 значень для побудови   
 # Гаусового розподілу  
p = norm.pdf(x, mu, sigma) # Отримання значень функції щільності

fig, ax = plt.subplots() # Створюємо порожній графік  
ax.plot(x, p, label='Гаус') # Додаємо дані до графіку  
ax.hist(ret, bins=50, # Побудова гістограми для прибутковостей  
 density=True,   
 alpha=0.6,   
 color='g',  
 label='прибутковості '+symbol)  
  
ax.legend() # Додаємо легенду  
ax.set\_xlabel("g") # Додаємо підпис для вісі Ох  
ax.set\_ylabel(r"$f(g)$") # Додаємо підпис для вісі Оу  
ax.set\_yscale('log')  
  
  
plt.savefig(f'Гаус + прибутковості {symbol}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 2.4: Порівняння функцій розподілу нормалізованих прибутковостей з нормальним розподілом |

Як ми можемо бачити, підігнана крива Гауса відхиляється від істинної частоти настання подій, що перевищують . Отже, ми можемо стверджувати, що наші прибутковості не є незалежними. Підтвердження цьому факту будемо шукати шляхом вивчення кореляційних властивостей нашого часового ряду.

Для простоти обчислень скористаємось функцією signal\_autocor() бібліотеки neurokit2. Виглядає дана функція наступним чином:

**signal\_autocor(signal, lag=None, demean=True, method='auto', show=False)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) - Вектор значень.
* **lag** (*int*) — Часовий лаг. Якщо вказано, буде повернуто одне значення автокореляції сигналу з його власним лагом.
* **demean** (*bool*) — Якщо має значення True, від сигналу буде відніматися середнє значення сигналу перед обчисленням автокореляції.
* **method** (*str*) — Використання "auto" запускає scipy.signal.correlate для використання швидшого алгоритму. Інші методи зберігаються з причин застарілості, але не рекомендуються. Інші методи включають "correlation" (за допомогою np.correlate()) або "fft" (швидке перетворення Фур’є).
* **show** (*bool*) — якщо значення True, побудувати графік автокореляції для всіх значень затримки.

**Повертає**

* **r** (*float*) - крос-кореляція сигналу з самим собою на різних часових лагах. Мінімальний часовий лаг дорівнює 0, максимальний часовий лаг дорівнює довжині сигналу. Або значення кореляції на певному часовому лазі, якщо лаг не дорівнює None.
* **info** (*dict*) - Словник, що містить додаткову інформацію, наприклад, довірчий інтервал.

# розрахунок автокореляції  
  
r\_init, \_ = nk.signal\_autocor(time\_ser.values,   
 method='correlation') # для вихідних значень ряду   
r\_ret, \_ = nk.signal\_autocor(ret,   
 method='correlation') # для прибутковостей  
r\_vol, \_ = nk.signal\_autocor(np.abs(ret),   
 method='correlation') # для модулів прибутковостей  
  
r\_range = np.arange(1, len(r\_ret) + 1) # генерація лагів

fig, ax = plt.subplots() # Створюємо порожній графік  
  
ax.plot(r\_range, r\_init[1:], label=symbol) # Додаємо дані до графіку  
ax.plot(r\_range, r\_ret, label=r'$g(t)$')   
ax.plot(r\_range, r\_vol, label=r'$V\_{T}$')   
  
ax.legend() # Додаємо легенду  
ax.set\_xlabel("Lag") # Додаємо підпис для вісі Ох  
ax.set\_ylabel("Autocorrelation r") # Додаємо підпис для вісі Оу  
ax.set\_ylim(-1.1, 1.1) # Встановлюємо обмеження по вісі Oy  
  
plt.savefig(f'Автокореляції {symbol}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 2.5: Зміна з часом парних автокореляційних функцій для вихідного ряду x, нормалізованих прибутковостей g та їх модулів mod(g) |

Але, досліджуючи складні системи варто пам’ятати, що їх складність є варіативною. Тому і внутрішні кореляції системи на різних часових лагах також варіюються з плином часу. Із цього випливає, що подальши розрахунки варто виконувати не для всього ряду, а для його фрагментів.

Подальші розрахунки здійснюватимуться в рамках алгоритму ковзного (рухомого) вікна. Для цього виділятиметься частина часового ряду (вікно), для якої розраховуватимуться міри складності, потім вікно зміщуватиметься разом з часовим рядом на заздалегідь визначену величину, і процедура повторюватиметься до тих пір, поки значення всього ряду не будуть вичерпані. Далі, порівнюючи динаміку фактичного часового ряду і відповідних мір складності, ми матимемо змогу судити про характерні зміни в динаміці міри складності зі зміною досліджуваної системи. Якщо та чи інша міра складності поводиться певним чином для всіх періодів крахів, наприклад, зменшується або збільшується під час передкризовий або передкритичний період, то вона може служити їх індикатором або провісником.

Розглянемо як поводитиме себе функція автокореляцій та волатильність в рамках алгоритму ковзного вікна.

Спочатку визначимо параметри

ret\_type = 4 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності),  
 # 6 - стандартизований вихідний ряд   
  
length = len(time\_ser) # довжина всього ряду  
  
window = 250 # довжина вікна - період у межах якого розраховуватимуться наші індикатори  
tstep = 1 # крок зміщення вікна  
volatility = [] # масив значень волатильностей   
autocorr = [] # масив значень автокореляції при змінній lag

Далі розпочнемо розрахунки. Для відслідковування прогресу зміщення ковзного вікна скористаємось бібліотекою tqdm. Її можна встановити аналогічно попереднім бібліотекам.

!pip install tqdm

Імпортуємо модуль для візуалізації прогресу

from tqdm import tqdm

і тепер приступимо до виконання віконної процедури:

for i in tqdm(range(0,length-window,tstep)): # Фрагменти довжиною window   
 # з кроком tstep  
   
 fragm = time\_ser.iloc[i:i+window].copy() # відбираємо фрагмент  
  
 # подальшому відбираємо потрібний тип ряду   
 if ret\_type == 1: # вихідні значення   
 pass  
 elif ret\_type == 2: # різниці  
 fragm = fragm[1:] - fragm[:-1]  
 elif ret\_type == 3: # прибутковості  
 fragm = fragm.pct\_change()  
 elif ret\_type == 4: # стандартизовані прибутковості  
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
 elif ret\_type == 5: # абсолютні значення прибутковостей  
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
 fragm = fragm.abs()  
 elif ret\_type == 6: # стандартизований вихідний ряд  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
   
 fragm = fragm.dropna().values # видаляємо зайві нульові значення, якщо є  
   
 # розрахунок віконної автокореляції  
 r\_window, \_ = nk.signal\_autocor(fragm, method='correlation')   
  
 # розрахунок волатильності по модулям прибутковостей   
 vol\_window = np.mean(np.abs(fragm))  
  
 # збереження результатів до масивів  
 volatility.append(vol\_window)  
 autocorr.append(r\_window[1])

100%|██████████| 1393/1393 [00:01<00:00, 884.25it/s]

Збережемо результати в окремих текстових файлах

# збереження результатів ковзної автокореляції  
np.savetxt(f"autocorr\_name={symbol}\_ \  
 window={window}\_step={tstep}\_ \  
 rettype={ret\_type}.txt", autocorr)  
  
# збереження результатів ковзної волатильності  
np.savetxt(f"volatility\_name={symbol}\_ \  
 window={window}\_step={tstep}\_ \  
 rettype={ret\_type}.txt", volatility)

Нарешті порівняємо динаміку вихідного ряду і розрахованих похідних. Для цього врахуємо, що автокореляцію і волатильність ми рахували для рухомого вікна. Результати представлено на рис. [Рисунок 2.6](#fig-btc-all).

fig, ax = plt.subplots(figsize=(13,8))  
  
ax2 = ax.twinx()  
ax3 = ax.twinx()  
ax4 = ax.twinx()  
  
ax2.spines.right.set\_position(("axes", 1.03))  
ax3.spines.right.set\_position(("axes", 1.10))  
ax4.spines.right.set\_position(("axes", 1.19))  
  
p1, = ax.plot(time\_ser.index[window:length:tstep],   
 time\_ser.values[window:length:tstep],   
 "b-", label=fr"{ylabel}")  
p2, = ax2.plot(time\_ser.index[window+1:length:tstep],   
 ret[window:length:tstep], "r--", label=r"$G(t)$")  
p3, = ax3.plot(time\_ser.index[window:length:tstep],   
 autocorr, "g-", label=r"$A$")  
p4, = ax4.plot(time\_ser.index[window:length:tstep],  
 volatility, "m-", label=r"$V$")  
  
ax.set\_xlabel(xlabel)  
ax.set\_ylabel(f"{ylabel}")  
  
ax.yaxis.label.set\_color(p1.get\_color())  
ax2.yaxis.label.set\_color(p2.get\_color())  
ax3.yaxis.label.set\_color(p3.get\_color())  
ax4.yaxis.label.set\_color(p4.get\_color())  
  
tkw = dict(size=4, width=1.5)  
ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
ax3.tick\_params(axis='y', colors=p3.get\_color(), \*\*tkw)  
ax4.tick\_params(axis='y', colors=p4.get\_color(), \*\*tkw)  
ax.tick\_params(axis='x', \*\*tkw)  
  
ax4.legend(handles=[p1, p2, p3, p4])  
  
plt.savefig(f"all\_name={symbol}\_ret={ret\_type}\_\  
 wind={window}\_step={tstep}.jpg")  
plt.show();

|  |
| --- |
| Рисунок 2.6: Динаміка індексу Біткоїна, нормалізованих прибутковостей, віконних автокореляції та волатильності |

Аналізуючи графік, можна зробити висновок, що у певні моменти спостерігалися стрибки волатильності (як і автокореляції) із поступовим зменшенням її до попереднього рівня, що може бути внаслідок збурень у процесі роботи ринку. Аналіз таких збурень, їх частоти та сили, дозволяє виявляти приховані закономірності роботи ринку.

## 2.3 Висновок

Таким чином, аналіз флуктуацій прибутковостей та волатильностей шляхом побудови функції автокореляції та розподілу ймовірності дозволяє отримати певні висновки, що можуть допомогти в роботі із аналізованими часовими рядами і ринком, з якого взято зазначені часові ряди. Зокрема, у даному випадку, можна давати рекомендації аналітикам, що працюють на фінансових ринках.

## 2.4 Завдання для самостійної роботи

1. Отримати індекс часового ряду у викладача
2. Провести дослідження згідно інструкції
3. Дослідити зміни розрахованих величин для вікон 100 і 500, з кроком 1. Порівняти результати
4. Зробити висновки

## 2.5 Контрольні питання

1. Порівняйте вид залежностей флуктуацій цін і прибутковостей. Чому при розрахунках користуються не цінами, а прибутковостями?
2. Яку характеристику ряду визначає волатильність?
3. У чому причина різних залежностей для прибутковостей та їх модулів?

## 2.6 Додаток

Для обрання часового індексу часового ряду використаємо дані, що розміщенні на сайті [Yahoo! Finance](https://finance.yahoo.com). Оскільки окремі фінансові показники не завжди є доступними, будемо використовувати список компаній, що входять до індексу [DJIA](https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average). За вказаним посиланням та номером у списку групи оберіть компанію, що входить до індексу та проведіть відповідні розрахунки. Порівняйте отримані результати з Біткоїном.

# 3. Лабораторна робота № 2

**Тема.** Використання рекурентного аналізу для моделювання і прогнозування нелінійних динамічних властивостей складних систем.

**Мета.** Навчитися інструментарію нелінійної динаміки, який відноситься до рекурентних властивостей нестаціонарних динамічних рядів.

## 3.1 Теоретичні відомості

Дослідження складних систем, як природних, так і штучних, показали, що в їх основі лежать нелінійні процеси, ретельне вивчення яких необхідне для розуміння і моделювання складних систем. У останні десятиліття набір традиційних (лінійних) методик дослідження був істотно розширений нелінійними методами, одержаними з теорії нелінійної динаміки і хаосу; багато досліджень були присвячені оцінці нелінійних характеристик і властивостей процесів, що протікають в природі (скейлінг, фрактальна розмірність). Проте більшість методів нелінійного аналізу вимагає або достатньо довгих, або стаціонарних рядів даних, які досить важко одержати з природи. Більш того, було показано, що дані методи дають задовільні результати для моделей реальних систем, що ідеалізуються. Ці чинники вимагали розробки нових методик нелінійного аналізу даних.

Стан природних або штучних систем, як правило, змінюється в часі. Вивчення цих, часто складних, процесів — важлива задача в багатьох дисциплінах, дозволяє зрозуміти і описати їх суть, наприклад, для прогнозування стану на деякий час в майбутнє. Метою таких досліджень є знаходження математичних моделей, які б достатньо відповідали реальним процесам і могли б бути використані для розв’язання поставлених задач.

Розглянемо ідею і коротко теорію рекурентного аналізу, наведемо деякі приклади, розглянемо його можливі області застосування при аналізі і прогнозування складних фінансово-економічних систем.

### 3.1.1 Фазовий простір та його реконструкція

Стан системи описується її змінними стану

де верхній індекс — номер змінної. Набір із змінних стану у момент часу складає вектор стану в -вимірному фазовому просторі. Даний вектор переміщається в часі в напрямі,визначуваному його вектором швидкості:

Послідовність векторів утворює траєкторію у фазовому просторі, причому поле швидкості дотичне до цієї траєкторії. Еволюція траєкторії описує динаміку системи і її атрактор. Знаючи , можна одержати інформацію про стан системи в момент шляхом інтегрування виразу. Оскільки форма траєкторії дозволяє судити про характер процесу (періодичні або хаотичні процеси мають характерні фазові портрети), то для визначення стану системи не обов’язково проводити інтегрування, достатньо побудувати графічне відображення траєкторії.

При дослідженні складних систем часто немає інформації про всі змінні стану, або не все з них можливо виміряти. Як правило, є єдине спостереження, проведене через дискретний часовий інтервал . Таким чином, вимірювання записуються у вигляді ряду i , де . Інтервал може бути постійним, проте це не завжди можливо і створює проблеми для застосування стандартних методів аналізу даних, що вимагають рівномірної шкали спостережень.

Взаємодії і їх кількість в складних системах такі, що навіть по одній змінній стану можна судити про динаміку всієї системи в цілому (даний факт був встановлений групою американських учених при вивченні турбулентності). Таким чином, еквівалентна фазова траєкторія, що зберігає структури оригінальної фазової траєкторії, може бути відновлена з одного спостереження або часового ряду за теоремою Такенса (Takens) методом часових затримок:

де — розмірність вкладення, — часова затримка (реальна часова затримка визначається як ). Топологічні структури відновленої траєкторії зберігаються, якщо , де — розмірність атрактора. На практиці більшості випадків атрактор може бути відновлений і при . Затримка, як правило, вибирається апріорно.

Існує кілька підходів до вибору мінімально достатньої розмірності , крім аналітичного. Високу ефективність показали методи, засновані на концепції фальшивих найближчих точок (false nearest neighbours, FNN). Суть її заключається у тому, що при зменшенні розмірності вкладення відбувається збільшення кількості фальшивих точок, що потрапляють в околицю будь-якої точки фазового простору. Звідси витікає простий метод — визначення кількості FNN як функції від розмірності. Існують і інші методи, засновані на цій концепції — наприклад, визначення відносин відстаней між одними і тими ж сусідніми точками при різних . Розмірність атрактора також може бути визначена за допомогою крос-кореляційних сум.

|  |  |
| --- | --- |
| |  | | --- | |  |   Рисунок 3.1: Відрізок траєкторії у фазовому просторі системи Рьослера (a); відповідний рекурентний графік (b). Вектор фазового простору в точці , який потрапляє в околицю (сіре коло в (a)) заданого вектора фазового простору вектора в точці вважається точкою рекурентності (чорна точка на траєкторії в (a)). Вона позначається чорною точкою на рекурентній діаграмі у позиції . Вектор фазового простору за межами околу (порожнє коло в (a)) призводить до білої точки в рекурентній діаграмі |

### 3.1.2 Рекурентний аналіз

Процесам в природі властива яскраво виражена рекурентна поведінка, така, як періодичність або іррегулярна циклічність. Більш того, рекурентність (повторюваність) станів в значенні проходження подальшої траєкторії достатньо близько до попередньої є фундаментальною властивістю дисипативних динамічних систем. Ця властивість була відмічена ще в 80-х роках XIX століття французьким математиком Пуанкаре (Poincare) і згодом сформульовано у вигляді “теореми рекурентності”, опублікованої в 1890 р.:

|  |
| --- |
| Примітка |
| **Якщо система зводить свою динаміку до обмеженої підмножини фазового простору, то система майже напевно, тобто з вірогідністю, практично рівною 1, скільки завгодно близько повертається до якого-небудь спочатку заданого режиму.** |

Суть цієї фундаментальної властивості у тому, що, не дивлячись на те, що навіть саме мале збурення в складній динамічній системі може привести систему до експоненціального відхилення від її стану, через деякий час система прагне повернутися до стану, деяким чином близького до попереднього, і проходить при цьому подібні етапи еволюції.

Переконатися в цьому можна за допомогою графічного зображення траєкторії системи у фазовому просторі. Проте можливості такого аналізу сильно обмежені. Як правило, розмірність фазового простору складної динамічної системи більша трьох, що робить практично незручним його розгляд напряму; єдина можливість — проекції в дво- і тривимірні простори, що часто не дає вірного уявлення про фазовий портрет.

У 1987 р. Екман (Eckmann) і співавтори запропонували спосіб відображення -вимірної фазової траєкторії станів системи завдовжки на двовимірну квадратну двійкову матрицю розміром , в якій 1 (чорна точка) відповідає повторенню стану при деякому часі в деякий інший час , а обидві координатні осі є осями часу. Таке представлення було назване рекурентною картою або діаграмою (recurrence plot, RP), оскільки воно фіксує інформацію про рекурентну поведінку системи.

Математично вищесказане описується як

де — кількість даних станів, — розмір околиці точки у момент , — норма і — функція Хевісайда.

Непрактично і, як правило, неможливо знайти повну рекурентність у значенні (стан динамічної, а особливо — хаотичної системи не повторюється повністю еквівалентно початковому стану, а підходить до нього скільки завгодно близько). Таким чином, рекурентність визначається як достатня близькість стану до стану . Іншими словами, рекурентними є стани , які потрапляють в -вимірну околицю з радіусом і центром в . Ці точки називаються **рекурентними точками** (recurrence points).

Оскільки , за визначенням, то рекурентна діаграма завжди міститьчорну діагональну лінію — лінію ідентичності (line of identity, LOI) під кутом до осей координат. Довільно узята рекурентна точка не несе якої-небудь корисної інформації про стани в часи і . Тільки вся сукупність рекурентних точок дозволяє відновити властивості системи.

Зовнішній вигляд рекурентної діаграми дозволяє судити про характер процесів, які протікають в системі, наявності і впливі шуму, станів повторення і завмирання (ламінарності), здійсненні в ході еволюції системи різких змін стану (екстремальних подій).

|  |
| --- |
| однорідна топологія дрейф Осцилююча поведінка системи Контрастна топологія Ламінарність процесу  Рисунок 3.2: Типові динамічні ряди і їх рекурентні карти |

### 3.1.3 Аналіз діаграм

Очевидно, що процеси різної поведінки даватимуть рекурентні діаграми з різним рисунком. Таким чином, візуальна оцінка діаграм може дати уявлення про еволюцію досліджуваної траєкторії. Виділяють два основних класи структури зображення: **топологія** (*typology*), що представляється крупномасштабними структурами, і **текстура** (*texture*), *що формується дрібномасштабними структурами*.

Топологія дає загальне уявлення про характер процесу. Виділяють чотири основні класи:

* **однорідні** рекурентні діаграми типові для стаціонарних і автономних систем, в яких час релаксації малий у порівнянні з довжиною ряду;
* **періодичні** структури, що повторюються (діагональні лінії, узори у шаховому порядку) відповідають різним осцилюючим системам з періодичністю в динаміці;
* **дрейф** відповідає системам з параметрами, що поволі змінюються, що робить білими лівий верхній і правий нижній кути рекурентної діаграми;
* **різкі зміни** в динаміці системи, рівно як і екстремальні ситуації, обумовлюють появу білих областей або смуг.

Рекурентні діаграми **спрощують** виявлення екстремальних і рідкісних подій.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  |   Рисунок 3.3: Характернi топологiї рекурентних дiаграм: (а) — однорiдна (нормально розподiлений шум); (b) — перiодична (генератор Ван дер Поля); (c) — дрейф (вiдображення Iкеди з накладеною послiдовнiстю, що лiнiйно росте); (d) — контрастнi областi або смуги (узагальнений броунiвський рух) |

Докладний розгляд рекурентних діаграм дозволяє виявити дрібномасштабні структури — текстуру, яка складається з простих точок, діагональних, горизонтальних і вертикальних ліній. Комбінації вертикальних і горизонтальних ліній формують прямокутні кластери точок.

* ***самотні***, окремо розташовані рекурентні точки з’являються в тому разі, коли відповідні стани рідкісні, або нестійкі в часі, або викликані сильною флуктуацією. При цьому вони не є ознаками випадковості або шуму;
* ***діагональні лінії*** (при де — довжина діагональної лінії) з’являються у разі, коли сегмент траєкторії у фазовому просторі пролягає паралельно іншому сегменту, тобто траєкторія повторює саму себе, повертаючись в одну і ту ж область фазового простору у різний час. Довжина таких ліній визначається часом, протягом якого сегменти траєкторії залишаються паралельними; напрям (кут нахилу) ліній характеризує внутрішній час підпроцесів, відповідних даним сегментам траєкторії. Проходження ліній паралельно лінії ідентичності (під кутом до осей координат) свідчить про однаковий напрям сегментів траєкторії, перпендикулярно — про протилежний («відображені» сегменти), що може також бути ознакою реконструкції фазового простору з невідповідною розмірністю вкладення. Нерегулярна поява діагональних ліній є ознакою хаотичного процесу;
* ***вертикальні (горизонтальні) лінії*** (при , де — довжина вертикальної або горизонтальної лінії) виділяють проміжки часу, в котрі стан системи не змінюється або змінюється трохи (система як би «заморожена» на цей час), що є ознакою «ламінарних» станів.

|  |
| --- |
| Рисунок 3.4: Основнi концепцiї рекурентного аналiзу. Вiдображена дiаграма рекурентностi базується на часовому ряду, що було реконструйовано до 11 реконструйованих векторiв, вiд до . Видiлено дiагональну лiнiю довжиною , вертикальна лiнiя довжиною i бiлу вертикальну лiнiю довжиною |

## 3.2 Хід роботи

Спочатку побудуємо дво- та тривимірні фазові портрети як для модельних значень, так і для реальних. Використовуватимемо бібліотеки neurokit2 для побудови атракторів та рекурентного аналізу.

### 3.2.1 Процедура реконструкції фазового простору

Для побудови фазового портрету скористаємось методами complexity\_attractor() та complexity\_embedding() бібліотеки neuralkit2. Синтаксис complexity\_attractor() виглядає наступним чином:

**complexity\_attractor(embedded='lorenz', alpha='time', color='last\_dim', shadows=True, linewidth=1, \*\*kwargs)**

**Параметри**

* **embedded** (*Union[str, np.ndarray]*) — результат функції complexity\_embedding(). Також може бути рядком, наприклад, "lorenz" (атрактор Лоренца) або "rossler" (атрактор Рьосслера).
* **alpha** (*Union[str, float]*) — прозорість ліній. Якщо "time", то лінії будуть прозорими як функція часу (повільно).
* **color** (*str*) — Колір графіку. Якщо "last\_dim", буде використано останній вимір (максимум 4-й) вбудованих даних, коли розмірність більша за 2. Корисно для візуалізації глибини (для 3-вимірного вбудовування), або четвертого виміру, але працюватиме це повільно.
* **shadows** (*bool*) — якщо значення True, 2D-проекції буде додано до бокових сторін 3D-атрактора.
* **linewidth** (*float*) — задає товщину лінії.
* **kwargs** — До палітри кольорів (наприклад, name="plasma") або до симулятора системи Лоренца передаються додаткові аргументи ключових слів, такі як duration (за замовчуванням = 100), sampling\_rate (за замовчуванням = 10), sigma (за замовчуванням = 10), beta (за замовчуванням = 8/3), rho (за замовчуванням = 28).

Як вже зазначалося, побудова фазового простору, на основі якого і проводитиметься рекурентний аналіз, вимагає реконструкції. Виконати реконструкції фазового простору із одновимірного часового ряду можна із використанням *методу часових затримок*.

Метод часових затримок є однією з ключових концепцій науки про складність, що ми використовуватимемо і в подальших лабораторних. Він базується на ідеї, що динамічна система може бути описана вектором чисел, який називається її “станом”, що має на меті забезпечити повний опис системи в певний момент часу. Множина всіх можливих станів називається “простором станів”.

Теорема Такенса (1981) припускає, що послідовність вимірювань динамічної системи містить у собі всю інформацію, необхідну для повної реконструкції простору станів. Метод часових затримок намагається визначити стан системи в певний момент часу , шукаючи в минулій історії спостережень схожі стани, і, вивчаючи еволюцію схожих станів, виводити інформацію про майбутнє системи.

Як візуалізувати динаміку системи? Послідовність значень стану в часі називається траєкторією. Залежно від системи, різні траєкторії можуть еволюціонувати до спільної підмножини простору станів, яка називається атрактором. Наявність та поведінка атракторів дає інтуїтивне уявлення про досліджувану динамічну систему.

Одже, згідно Такенсу, ідея полягає в тому, щоб на основі одиничних вимірювань системи, отримати -розмірні реконструйовані часові вкладення

де проходить в діапазоні ; значення представляє часову затримку, а — це розмірність вкладень (кількість змінних, що включає кожна траєкторія).

Код для реконструкції фазового простору може виглядати наступним чином:

def complexity\_embedding(signal, dimension, delay):  
 N = len(signal) # вимірюємо довжину сигналу  
 Y = np.zeros((dimension, N - (dimension - 1) \* delay)) # ініціалізуємо масив нулів,  
 # що будуть представляти траєкторії  
 for i in range(dimension):  
 Y[i] = signal[i \* delay : i \* delay + Y.shape[1]] # заповнюємо кожну траєкторію   
  
 embedded = Y.T   
 return embedded # повертаємо результат

Для реконструкції фазового простору використовуватимемо метод complexity\_embedding(). Його синтаксис виглядає наступним чином:

**complexity\_embedding(signal, delay=1, dimension=3, show=False, \*\*kwargs)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень. Також може бути рядком, наприклад, "lorenz" (атрактор Лоренца), "rossler" (атрактор Росслера) або "clifford" (атрактор Кліффорда) для отримання попередньо визначеного атрактора.
* **delay** (*int*) — часова затримка (часто позначається іноді називають запізненням). Ще розглянемо метод complexity\_delay() для оцінки оптимального значення цього параметра.
* **dimension** (*int*) — розмірність вкладень (, іноді позначається як або порядок). Далі звернемось до методу complexity\_dimension(), щоб оцінити оптимальне значення для цього параметра.
* **show** (*bool*) — Побудувати графік реконструйованого атрактора.
* **kwargs** — інші аргументи, що передаються до complexity\_attractor().

**Повертає**

* *array* — реконструйований атрактор розміру length - (dimension - 1) \* delay

Далі імпортуємо необхідні для подальшої роботи модулі

import matplotlib.pyplot as plt   
import numpy as np  
import neurokit2 as nk  
import yfinance as yf  
import scienceplots  
  
%matplotlib inline

І виконаємо налаштування рисунків для виводу

plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься  
 # для виведення рисунків  
  
params = {  
 'figure.figsize': (8, 6), # встановлюємо ширину та висоту рисунків за замовчуванням  
 'font.size': 22, # розмір фонтів рисунку  
 'lines.linewidth': 2, # товщина ліній  
 'axes.titlesize': 'small', # розмір титулки над рисунком  
 "font.family": "sans-serif", # сімейство стилів підписів   
 "font.serif": ["Times"], # стиль підпису  
 'savefig.dpi': 300 # якість збережених зображень  
}  
  
plt.rcParams.update(params) # оновлення стилю згідно налаштувань

Тепер розглянемо можливість використання методу часових затримок і отриманих в подальшому атракторів у якості індикатора складності. Як і в попередній роботі, для прикладу завантажимо часовий ряд Біткоїна за період з 1 вересня 2015 по 1 березня 2020, використовуючи yfinance:

symbol = 'BTC-USD' # Символ індексу  
start = "2015-09-01" # Дата початку зчитування даних  
end = "2020-03-01" # Дата закінчення зчитування даних  
  
data = yf.download(symbol, start, end) # вивантажуємо дані  
time\_ser = data['Adj Close'].copy() # зберігаємо саме ціни закриття

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

Спочатку оберемо вид ряду: 1. вихідний ряд 2. детермінований (різниця між теперішнім та попереднім значенням) 3. прибутковості звичайні 4. стандартизовані прибутковості 5. абсолютні значення (волатильності) 6. стандартизований ряд

Для подальших розрахунків накращим варіантом буде вибір стандартизованого вихідного ряду або прибутковостей, оскільки значення вихідного часового ряду відрізняються на декілька порядків, і можуть сильно перевищувати встановлений параметр . Тобто, для вихідних значень, що сильно різняться між собою, увесь часовий діапазон буде розглядатися як нерекурентний.

Спочатку визначимо функції для виконання перетворення ряду:

def transformation(signal, ret\_type):  
  
 for\_rec = signal.copy()  
  
 if ret\_type == 1: # Зважаючи на вид ряду, виконуємо  
 # необхідні перетворення  
 pass  
 elif ret\_type == 2:  
 for\_rec = for\_rec.diff()  
 elif ret\_type == 3:  
 for\_rec = for\_rec.pct\_change()  
 elif ret\_type == 4:  
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 elif ret\_type == 5:   
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 for\_rec = for\_rec.abs()  
 elif ret\_type == 6:  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
  
 for\_rec = for\_rec.dropna().values  
  
 return for\_rec

і тепер виконаємо перетворення, використовуючи дану функцію:

signal = time\_ser.copy()  
ret\_type = 6 # вид ряду: 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
for\_rec = transformation(signal, ret\_type)

Оскільки ми не матимемо змоги візуалізувати багатовимірний фазовий простір (), ми послуговуватимемось значеннями та . Значення будемо варіювати як із власних переконань, так і з опорою на функціонал бібліотеки neuralkit2.

Скористаємось методом complexity\_simulate() для генерації різних тестових сигналів.

signal\_random\_walk = nk.complexity\_simulate(duration=30,   
 sampling\_rate=100,   
 method="randomwalk") # симуляція випадкового блукання

nk.complexity\_attractor(embedded=nk.complexity\_embedding(signal\_random\_walk, dimension=2, delay=100),   
 alpha=1,   
 color="orange");

|  |
| --- |
| Рисунок 3.5: Двовимірний фазовий портрет випадкового блукання |

nk.complexity\_attractor(nk.complexity\_embedding(signal\_random\_walk, dimension=3, delay=100),   
 alpha=1,   
 color="orange");

|  |
| --- |
| Рисунок 3.6: Тривимірний фазовий портрет випадкового блукання |

signal\_ornstein = nk.complexity\_simulate(duration=30,   
 sampling\_rate=100,   
 method="ornstein") # симуляція системи Орнштайна

nk.complexity\_attractor(nk.complexity\_embedding(signal\_ornstein, dimension=2, delay=100),   
 alpha=1,   
 color="red");

|  |
| --- |
| Рисунок 3.7: Двовимірний фазовий портрет системи Орнштайна |

nk.complexity\_attractor(nk.complexity\_embedding(signal\_ornstein, dimension=3, delay=100),   
 alpha=1,   
 color="red");

|  |
| --- |
| Рисунок 3.8: Двовимірний фазовий портрет системи Орнштайна |

nk.complexity\_attractor(color = "last\_dim", alpha="time", duration=1);

|  |
| --- |
| Рисунок 3.9: Тривимірний фазовий портрет атрактора Лоренца |

nk.complexity\_attractor("rossler", color = "blue", alpha=1, sampling\_rate=5000);

|  |
| --- |
| Рисунок 3.10: Тривимірний фазовий портрет атрактора Рьосслера |

nk.complexity\_attractor(nk.complexity\_embedding(for\_rec, dimension=2, delay=100),   
 alpha=1,   
 color="lime");

|  |
| --- |
| Рисунок 3.11: Двовимірний фазовий портрет вихідних значень досліджуваного ряду Біткоїна |

nk.complexity\_attractor(nk.complexity\_embedding(for\_rec, dimension=3, delay=100),   
 alpha=1,   
 color="lime");

|  |
| --- |
| Рисунок 3.12: Тривимірний фазовий портрет вихідних значень досліджуваного ряду Біткоїна |

У зазначених вище прикладах прикладах ми обирали параметри і згідно нашим власним міркуванням. Але, як правило, при виконанні серйозного дослідження, що матиме прикладне застосування, лише власних переконань буває недостатньо. У нашому випадку бажано було б, щоб зазначені параметри обирались автоматично, опираючись на конкретну статистичну процедуру. Бібліотека neurokit2 представляє функціонал для автоматичного підбору параметрів розмірності та часової затримки. Коротко їх опишемо.

### 3.2.2 Автоматизований підбір параметра часової затримки,

Часова затримка (*Tau* також відома як *Lag*) є одним з двох критичних параметрів, що беруть участь у процедурі реконструкції фазового простору. Він відповідає затримці у відліках між вихідним сигналом і його затриманою версією (версіями). Іншими словами, скільки відліків ми розглядаємо між певним станом сигналу та його найближчим минулим станом.

Якщо менше оптимального теоретичного значення, послідовні координати стану системи корельовані і атрактор недостатньо розгорнутий. І навпаки, коли більше, ніж повинно бути, послідовні координати майже незалежні, що призводить до некорельованої та неструктурованої хмари точок.

Вибір параметрів *затримки* та *розмірності* представляє нетривіальну задачу. Один з підходів полягає у їх (напів)незалежному виборі (оскільки вибір розмірності часто вимагає затримки) за допомогою функцій complexity\_delay() та complexity\_dimension(). Однак, існують методи спільного оцінювання, які намагаються знайти оптимальну затримку та розмірність одночасно.

Зауважте також, що деякі автори (наприклад, Розенштейн, 1994) пропонують спочатку визначити оптимальну розмірність вбудовування, а потім розглядати оптимальне значення затримки як оптимальну затримку між першою та останньою координатами затримки (іншими словами, фактична затримка має дорівнювати оптимальній затримці, поділеній на оптимальну розмірність вбудовування мінус 1).

Декілька авторів запропонували різні методи для вибору затримки:

* **Фрейзер і Свінні (1986)** пропонують використовувати перший локальний мінімум взаємної інформації між затриманим і незатриманим часовими рядами, ефективно визначаючи значення , для якого вони діляться найменшою інформацією (і де атрактор є найменш надлишковим). На відміну від автокореляції, взаємна інформація враховує також нелінійні кореляції.
* **Тейлер (1990)** запропонував вибирати таке значення , при якому автокореляція між сигналом та його зміщенною версією при вперше перетинає значення . Методи, що базуються на автокореляції, мають перевагу в короткому часі обчислень, коли вони обчислюються за допомогою алгоритму швидкого перетворення Фур’є (fast Fourier transform, FFT).
* **Касдаглі (1991)** пропонує замість цього брати перший нульовий перетин автокореляції.
* **Розенштейн (1993)** пропонує апроксимувати точку, де функція автокореляцій падає до від свого максимального значення.
* **Розенштейн (1994)** пропонує наближатися до точки, близької до 40% нахилу середнього зміщення від діагоналі.
* **Кім (1999)** пропонує оцінювати Tau за допомогою кореляційного інтегралу, який називається C-C методом, і який, як виявилося, узгоджується з результатами, отриманими за допомогою методу взаємної інформації. Цей метод використовує статистику в реконструйованому фазовому просторі, а не аналізує часову еволюцію ряду. Однак час обчислень для цього методу значно довший через необхідність порівнювати кожну унікальну пару парних векторів у реконструйованому сигналі на кожну затримку.
* **Лайл (2021)** описує “Реконструкцію симетричного проекційного атрактора” (Symmetric Projection Attractor Reconstruction, SPAR), де від домінуючої частоти (тобто довжини середнього “циклу”) може бути підходящим значенням для приблизно періодичних даних, і робить атрактор чутливим до морфологічних змін. Див. також [доповідь Астона](https://youtu.be/GGrOJtcTcHA?t=730). Цей метод також є найшвидшим, але може не підходити для аперіодичних сигналів. Аргумент algorithm (за замовчуванням "fft") передається до аргументу method методу [signal\_psd()](https://neuropsychology.github.io/NeuroKit/functions/signal.html#signal-psd).

Можна також зазначити наступний метод для об’єднаного підбору параметрів затримки та розмірності:

* **Гаутама (2003)** зазначає, що на практиці часто використовують фіксовану часову затримку і відповідно регулюють розмірність вбудовування. Оскільки це може призвести до великих значень (а отже, до вкладених даних великого розміру) і, відповідно, до повільної обробки, вони описують метод оптимізації для спільного визначення і на основі показника **entropy ratio**.

Розглянемо оптимальні значення розмірності та затримки для часового сигналу Біткоїна:

delay, parameters = nk.complexity\_delay(for\_rec,   
 delay\_max=300, show=True,  
 method="fraser1986")

|  |
| --- |
| Рисунок 3.13: Оптимальне значення часової затримки на основі методу Фрейзера і Свінні для часового ряду Біткоїна |

[Рисунок 3.13](#fig-btc-delay-fraser1986) показує, що перший локальний мінімум взаємної інформації для стандартизованих вихідних значень Біткоїна знаходиться на 273 лагу. Для візуального огляду реконструйованого атрактора це значення, можливо, є найбільш адекватним. Але використовуючи настільки велику часову затримку, ми втрачаємо доволі багато проміжних значень, що також можуть містити досить важливу приховану інформацію для кількісних розрахунків.

delay, parameters = nk.complexity\_delay(for\_rec,   
 delay\_max=300, show=True,  
 method="theiler1990")

|  |
| --- |
| Рисунок 3.14: Оптимальне значення часової затримки на основі методу Тейлера для часового ряду Біткоїна |

[Рисунок 3.14](#fig-btc-delay-theiler) демонструє, що автокореляція між стандартизованих вихідним сигналом Біткоїна та його зміщенною версією при вперше перетинає значення . Бачимо, що дане значення затримки є трохи меншим за те, що було отримано до цього, але суті це не змінює. Також бачимо, що між реконструйованими атракторами для та немає кардинальної візуальної різниці.

delay, parameters = nk.complexity\_delay(for\_rec,   
 delay\_max=500, show=True,  
 method="casdagli1991")  
  
delay

Як можна бачити по прикладу вище, не всі методи надають адекватну оцінку розмірності нашого сигналу. Спробуємо привести вихідні значення Біткоїна до прибутковостей та повторити процедуру Касдаглі ще раз.

ret\_type = 4   
ret = transformation(signal, ret\_type)

delay, parameters = nk.complexity\_delay(ret,   
 delay\_max=300, show=True,  
 method="casdagli1991")

|  |
| --- |
| Рисунок 3.15: Оптимальне значення часової затримки на основі методу Касдаглі для прибутковостей Біткоїна |

Цього разу нам вдалося досягти оптимального результату, але приклад вище демонструє, що кожна процедура має свої виключення. [Рисунок 3.15](#fig-btc-delay-casdagli1991) показує, що значення прибутковостей Біткоїна характеризуються певними кореляціями лише на перших 4-ох лагах. Подальші часові зміщення роблять значення прибутковостей незалежними один від одного.

delay, parameters = nk.complexity\_delay(for\_rec,   
 delay\_max=300, show=True,  
 method="rosenstein1993")

|  |
| --- |
| Рисунок 3.16: Оптимальне значення часової затримки на основі методу Розенштайна (1993) для часового ряду Біткоїна |

[Рисунок 3.16](#fig-btc-delay-rosenstein1993) демонструє, що при функція автокореляцій перетинає значення . При цьому видно, що навіть для такого лагу зберігається значна частка кореляцій між стандартизованими вихідними значеннями Біткоїна.

delay, parameters = nk.complexity\_delay(for\_rec,   
 delay\_max=300, show=True,  
 method="rosenstein1994")

|  |
| --- |
| Рисунок 3.17: Оптимальне значення часової затримки на основі методу Розенштайна (1994) для часового ряду Біткоїна |

Рисунок вище показує, що при зміщення реконструйованих траєкторій від їх оригінального положення на лінії ідентичності зберігає найбільшу кількість інформації стосовно атрактора стандартизованих значень Біткоїна.

delay, parameters = nk.complexity\_delay(for\_rec,   
 delay\_max=300, show=True,  
 method="lyle2021")

|  |
| --- |
| Рисунок 3.18: Оптимальне значення часової затримки на основі методу Лайла для часового ряду Біткоїна |

Згідно представленого вище результату найбільш значущі частоти, отримані за допомогою перетворення Фур’є, зберігаються при .

Тепер подивимось як це виглядатиме для об’єднаного підбору параметрів

delay, parameters = nk.complexity\_delay(for\_rec,  
 delay\_max=np.arange(1, 30, 1), # діапазон значень затримки  
 dimension\_max=20, # максимальна розмірність вкладень  
 method="gautama2003",  
 surrogate\_n=5, # Кількість сурогатних сигналів   
 # для генерації  
 surrogate\_method="random", # Спосіб генерації сигналів  
 show=True)

|  |
| --- |
| Рисунок 3.19: Оптимальне значення розмірності та затримки на основі методу Гаутами для часового ряду Біткоїна |

dimension = parameters["Dimension"]  
dimension

20

Оскільки представлена вище процедура є доволі громіздкою в плані обчислювальних потужностей, ми обрали діапазон в межах від 1 до 30. Видно, що при близької до 10 оптимальне значення розмірності атрактора дорівнює 20. Можливо, при значеннях близьких до 100 або 200, ми могли б отримати зовсім інше значення розмірності, але це потребує додаткових експериментів.

### 3.2.3 Автоматизований підбір параметра розмірності вкладень,

За дану процедуру відповідає метод complexity dimension(). Її синтаксис виглядає наступним чином:

**complexity\_dimension(signal, delay=1, dimension\_max=20, method='afnn', show=False, \*\*kwargs)**

Хоча зазвичай використовують або , але різні автори пропонують наступні процедури підбору:

* **Кореляційна розмірність (Correlation Dimension, CD)**: Одним з перших методів оцінки оптимального був розрахунок кореляційної розмірності для вкладень різного розміру і пошук насичення (тобто плато) в її значенні при збільшенні розміру векторів. Одне з обмежень полягає в тому, що насичення буде також мати місце, коли даних недостатньо для адекватного заповнення простору високої розмірності (зауважте, що в загальному випадку не рекомендується мати настільки великі вкладення, оскільки це значно скорочує довжину сигналу).
* **Найближчі хибні сусіди (False Nearest Neighbour, FNN)**: Метод, запропонований Кеннелом та ін., базується на припущенні, що дві точки, які є близькими одна до одної в достатній розмірності вбудовування, повинні залишатися близькими при збільшенні розмірності. Алгоритм перевіряє сусідів при збільшенні розмірності вкладень, поки не знайде лише незначну кількість хибних сусідів при переході від розмірності до . Це відповідає найнижчій розмірності вкладення, яка, як передбачається, дає розгорнуту реконструкцію просторово-часового стану. Цей метод може не спрацювати в зашумлених сигналах через марну спробу розгорнути шум (а в чисто випадкових сигналах кількість хибних сусідів суттєво не зменшується зі збільшенням ). На рисунку нижче показано, як проекції на простори більшої розмірності можна використовувати для виявлення хибних найближчих сусідів. Наприклад, червона та жовта точки є сусідами в одновимірному просторі, але не в двовимірному.

|  |
| --- |
|  |

* **Середні хибні сусіди (Average False Neighbors, AFN)**: Ця модифікація методу FNN, розроблена Сао (1997), усуває один з його основних недоліків — необхідність евристичного вибору порогових значень . Метод використовує максимальну евклідову відстань для представлення найближчих сусідів і усереднює всі відношення відстані в розмірності до розмірності і визначає *E1* та *E2* як параметри. Оптимальна розмірність відповідає досягається тоді, коли *E1* перестає змінюватися (досягає плато). *E1* досягає плато при розмірності *d0*, якщо сигнал надходить від атрактора. Тоді *d0*+1\* є оптимальною мінімальною розмірністю вкладення. *E2* є корисною величиною для того, щоб відрізнити детерміновані сигнали від стохастичних. Константа *E2*, що близька до 1 для будь-якої розмірності вкладень , вказує на випадковість даних, оскільки майбутні значення не залежать від минулих значень.

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **delay** (*int*) — часова затримка у відліках. Для вибору оптимального значення цього параметра ми ще скористаємось методом complexity\_delay().
* **dimension\_max** (*int*) — максимальний розмір вкладення для тестування.
* **method** (*str*) — Може бути "afn" (середні хибні сусіди), "fnn" (найближчий хибний сусід) або "cd" (кореляційна розмірність).
* **show** (*bool*) — Візуалізувати результат.
* **kwargs** — інші аргументи, такі як або (відносне та абсолютне граничне значення, тільки для методу "fnn").

**Повертає**

* **dimension** (*int*) — оптимальна розмірність вкладень.
* **parameters** (*dict*) — словник python, що містить додаткову інформацію про параметри, які використовуються для обчислення оптимальної розмірності.

Спробуємо отримати оптимальне значення розмірності згідно зазначених процедур. В якості часової затримки можна взять . Приблизно таке значення спостерігалося для кожної процедури.

optimal\_dimension, info = nk.complexity\_dimension(for\_rec,  
 delay=100,  
 dimension\_max=10,  
 method='cd',  
 show=True)

|  |
| --- |
| Рисунок 3.20: Оптимальне значення розмірності на основі кореляційної розмірності для часового ряду Біткоїна |

[Рисунок 3.20](#fig-btc-dim-cd) представляє, що оптимальна розмірність вкладень при якій досягається найбільш інформативна репрезентація фазового простору дорівнює 7.

optimal\_dimension, info = nk.complexity\_dimension(for\_rec,  
 delay=100,  
 dimension\_max=10,  
 method='fnn',  
 show=True)

|  |
| --- |
| Рисунок 3.21: Оптимальне значення розмірності на основі найближчих хибних сусідів для часового ряду Біткоїна |

З представленого вище рисунку видно, що найнижча розмірності вкладення, яка, як передбачається, дає розгорнуту реконструкцію просторово-часового стану, дорівнює 3. Саме при переході від 3-ох вимірного фазового простору до 4-ох вимірного ми бачимо, що кількість хибних сусідів стає мінімальною і далі не наростає.

optimal\_dimension, info = nk.complexity\_dimension(for\_rec,  
 delay=20,  
 dimension\_max=20,  
 method='afnn',  
 show=True)

|  |
| --- |
| Рисунок 3.22: Оптимальне значення розмірності на основі середніх найближчих хибних сусідів для часового ряду Біткоїна |

Алгоритм середніх хибних сусідів показує, що тут розмірність вкладень є найоптимальнішою. При подальшому наростанні розмірності, атрактор має походити на більш стохастичний, що вказує на втрату всіх кореляцій, що могли бути присутні в досліджуваному сигналі.

Згідно з представленими вище алгоритмами автоматичного підбору, розмірність вкладень можна обирати в діапазоні значень від 3 до 7. Тепер на основі отриманих результатів приступимо до побудови рекурентної діаграми.

### 3.2.4 Побудова рекурентної матриці

Як вже зазначалося, рекурентний аналіз кількісно визначає кількість і тривалість рекурентних станів динамічної системи, що визначаються на основі реконструйованих траєкторій фазового простору.

Ми маємо змогу побудувати рекурентну матрицю, використовуючи метод recurrence\_matrix().

Його синтаксис виглядає наступним чином:

**recurrence\_matrix(signal, delay=1, dimension=3, tolerance='default', show=False)**

**Параметри**

* **signal** (*Union[list, np.ndarray, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **delay** (*int*) — затримка в часі.
* **dimension** (*int*) — розмірність вкладень, .
* **tolerance** (*float*) — радіус багатовимірного околу в межах якого шукаються рекурентні траєкторії (часто позначається як ), відстань, на якій дві точки даних вважаються схожими. Якщо "sd" (за замовчуванням), буде встановлено значення . Емпіричним правилом є встановлення таким чином, щоб відсоток точок, класифікованих як рекурентні, становив приблизно 2-5%.
* **show** (*bool*) — візуалізувати рекурентну матрицю.

**Повертає**

* *np.ndarray* — рекурентну матрицю.
* *np.ndarray* — матрицю відстаней.

Побудуємо рекурентну матрицю для вихідних значень Біткоїна, його прибутковостей та стандартизованого вихідного ряду. Розмірність , часова затримка , радіус .

rc, \_ = nk.recurrence\_matrix(signal,   
 delay=1,   
 dimension=4,   
 tolerance=0.3,  
 show=True)

|  |
| --- |
| Рисунок 3.23: Рекурентна матриця для вихідних значень Біткоїна |

Як можна бачити з представленого рисунку всі траєкторії залишаються доволі віддаленими один від одного, ніякої рекурентності тут не передбачається.

Тепер спробуємо подивитися на стандартизовані прибутковості.

rc, \_ = nk.recurrence\_matrix(ret,   
 delay=1,   
 dimension=4,  
 tolerance=0.3,  
 show=True)

|  |
| --- |
| Рисунок 3.24: Рекурентна матриця для стандартизованих прибутковостей Біткоїна |

Тепер можемо бачити, що Біткоїн став характризуватися чорними смугами, що відображають динаміку певних детермінованих процесів. У той же час білі смуги характеризують періоди абсолютно аномальної (непередбачуваної поведінки на даному ринку). Видно, що прибутковості залишаються доволі некорельованими, про що і свідчить переважне домінування саме білих областей.

Спробуємо тепер подивитись на стандартизований вихідний ряд.

rc, \_ = nk.recurrence\_matrix(for\_rec,   
 delay=1,   
 dimension=4,  
 tolerance=0.3,  
 show=True)

|  |
| --- |
| Рисунок 3.25: Рекурентна матриця для стандартизованого вихідного ряду Біткоїна |

На початку свого існування Біткоїн характеризувався доволі високим ступенем передбачуваності, меншої волатильності власних коливань. Надалі почали предомінувати білі області, але видно, що тепер Біткоїну властива динаміка подібна до броунівсього руху.

## 3.3 Завдання для самостійної роботи

1. Отримати індекс часового ряду у викладача
2. Провести дослідження його рекурентних властивостей згідно інструкції
3. Порівняти фазові портрети і рекурентні діаграми для стандартизованого вихідного ряду та прибутковостей. Що спільного між ними і чим вони відрізняються?
4. Зробити висновки

# 4. Лабораторна робота № 3

**Тема.** Кількісний аналіз рекурентних діаграм.

**Мета.** Ознайомитись з кількісними оцінками рекурентних діаграм для аналізу динаміки складних систем.

## 4.1 Теоретичні відомості

Для якісного опису системи графічне представлення системи підходить якнайкраще. Однак головним недоліком графічного представлення є те, що воно змушує користувачів суб’єктивно інтуїтивно інтерпретувати закономірності та структури, представлені на рекурентній діаграмі.

Крім того, зі збільшенням розміру даних, проблематичним представляється аналіз усіх значень. Як наслідок, доводиться працювати з окремими ділянками вихідних даних. Аналіз у такий спосіб може створювати нові дефекти, які спотворюють об’єктивність спостережуваних закономірностей і призводять до неправильних інтерпретацій. Щоб подолати це обмеження і поширити об’єктивну оцінку серед дослідників, на початку 1990-х років Веббером та Збілутом були введені визначення та процедури для кількісної оцінки складності рекурентних діаграм, а згодом вони були розширені Марваном та ін.

Дрібномасштабні кластери можуть являти собою комбінацію ізольованих точок (випадкових рекурентностей). Подібна еволюція в різні періоди часу або в зворотному часовому порядку представлятиме діагональні лінії (детерміновані структури), а також вертикальні/горизонтальні лінії для позначення ламінарних станів (переривчастість) або станів, що предсталяють сингулярності. Для кількісного опису системи системи такі дрібномасштабні кластери слугують основою кількісного рекурентного аналізу (recurrence quantification analysis, RQA).

## 4.2 Хід роботи

Перш ніж переходити до опису кожної з мір та розрахунків, визначемось з інструментарієм для виконання RQA. Як і до цього, ми використовуватимемо бібліотеку neuralkit2.

Тепер імпортуємо бібліотеки для подальшої роботи:

import matplotlib.pyplot as plt   
import numpy as np  
import neurokit2 as nk  
import yfinance as yf  
import scienceplots  
from tqdm import tqdm  
  
%matplotlib inline

І виконаємо налаштування рисунків для виводу:

plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься  
 # для виведення рисунків  
  
params = {  
 'figure.figsize': (8, 6), # встановлюємо ширину та висоту рисунків за замовчуванням  
 'font.size': 22, # розмір фонтів рисунку  
 'lines.linewidth': 2, # товщина ліній  
 'axes.titlesize': 'small', # розмір титулки над рисунком  
 "font.family": "sans-serif", # сімейство стилів підписів   
 "font.serif": ["Times"], # стиль підпису  
 'savefig.dpi': 300 # якість збережених зображень  
}  
  
plt.rcParams.update(params) # оновлення стилю згідно налаштувань

Розглянемо можливість використання всіх згаданих показників у якості індикаторів або індикаторів-передвісників кризових явищ. Для прикладу завантажимо часовий ряд фондового індексу Доу-Джонса за період з 1 грудня 1993 по 1 грудня 2022, використовуючи yfinance:

symbol = '^DJI' # Символ індексу  
start = "1993-01-01" # Дата початку зчитування даних  
end = "2022-01-01" # Дата закінчення зчитування даних  
  
data = yf.download(symbol, start, end) # вивантажуємо дані  
time\_ser = data['Adj Close'].copy() # зберігаємо саме ціни закриття  
  
xlabel = 'time, days' # підпис по вісі Ох   
ylabel = symbol # підпис по вісі Оу

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

Виведемо досліджуваний ряд:

fig, ax = plt.subplots() # Створюємо порожній графік  
ax.plot(time\_ser.index, time\_ser.values) # Додаємо дані до графіку  
ax.legend([symbol]) # Додаємо легенду  
ax.set\_xlabel(xlabel) # Встановимо підпис по вісі Ох  
ax.set\_ylabel(ylabel) # Встановимо підпис по вісі Oy  
  
plt.xticks(rotation=45) # оберт позначок по осі Ох на 45 градусів  
  
plt.savefig(f'{symbol}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 4.1: Динаміка щоденних змін індексу Доу-Джонса |

Користуючись тими методами, що ми розглянули в попередній лабораторній роботі, побудуємо атрактор даного ряда та його рекурентну діаграму. Але перш за все, треба стандартизувати наш ряд. Для цього оголосимо функцію transformation(), що прийматиме на вхід часовий сигнал, тип ряду, і повертатиме його перетворення:

def transformation(signal, ret\_type):  
  
 for\_rec = signal.copy()  
  
 if ret\_type == 1: # Зважаючи на вид ряду, виконуємо  
 # необхідні перетворення  
 pass  
 elif ret\_type == 2:  
 for\_rec = for\_rec.diff()  
 elif ret\_type == 3:  
 for\_rec = for\_rec.pct\_change()  
 elif ret\_type == 4:  
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 elif ret\_type == 5:   
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 for\_rec = for\_rec.abs()  
 elif ret\_type == 6:  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
  
 for\_rec = for\_rec.dropna().values  
  
 return for\_rec

Далі приводимо ряд до стандартизованого вигляду.

signal = time\_ser.copy()  
ret\_type = 6 # вид ряду: 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
for\_rec = transformation(signal, ret\_type)

Для всього ряду і для віконної процедру визначимо наступні параметри:

* розмірність вкладень ;
* часова затримка ;
* радіус багатовимірного околу .

Задамо необхідні параметри для обчислення та виводу:

m = 3 # розмірність вкладень  
tau = 1 # часові затримка  
eps = 0.3 # радіус

І тепер подивимось на фазові траєкторії досліджуваної системи у дво- та тривимірному просторах:

nk.complexity\_attractor(nk.complexity\_embedding(for\_rec, dimension=2, delay=tau),   
 alpha=1,   
 color="red");

|  |
| --- |
| Рисунок 4.2: Двовимірний фазовий портрет стандартизованих вихідних значень досліджуваного ряду Доу-Джонса |

nk.complexity\_attractor(nk.complexity\_embedding(for\_rec, dimension=3, delay=tau),   
 alpha=1,   
 color="red");

|  |
| --- |
| Рисунок 4.3: Тривимірний фазовий портрет стандартизованих вихідних значень досліджуваного ряду Доу-Джонса |

Як можна бачити по візуальному огляду траєкторій у фазовому просторі важко робити висновки стосовно передбачуванності або хаотичності системи. Спробуємо ще раз, але тепер послуговуючись рекурентною діаграмою:

rc, \_ = nk.recurrence\_matrix(for\_rec,   
 delay=1,   
 dimension=m,  
 tolerance=eps,  
 show=True)

|  |
| --- |
| Рисунок 4.4: Рекурентна матриця для стандартизованого вихідного ряду Доу-Джонса |

Як можна бачити, на основі рекурентної діаграми в перспективі ми можемо отримати куди більше інформації стосовно еволюції системи. Видно, що 2000-2008 рік характеризувалися найвищим ступенем самоорганізації (рекурентності) про що свідчать доволі велика щільність чорних областей. У той же час можна бачити, що останні роки характеризуються найменшим ступенем рекурентності. Можливо, прогнозованість подій у межах 2022 року варто було б охарактеризувати за допомогою інших індикаторів, але але рекурентна матриця говорить, що події минулих років мало корелюють з теперішнім.

Ми вже зазначали, що якісна репрезентація рекурентності станів не є достатньо об’єктивною. Найращим варіантом у даному випадку буде використання рекурентного аналізу наряду с алгоритмом рухомого вікна, що використовувався нами у першій лабораторній роботі, і буде використовуватись і надалі.

### 4.2.1 Віконна процедура

Для подальшої роботи створюємо віконну процедуру, в якій знов визначаємо вид ряду та ще декілька параметрів. Потім ми ініціалізуємо масиви для кожної рекурентної міри.

ret\_type = 6 # вид ряду  
window = 250 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(time\_ser) # довжина самого ряду  
  
m = 1 # розмірність вкладень  
tau = 1 # часові затримка  
eps = 0.3 # радіус  
  
 # Ініціалізуємо масиви для збереження віконних значень   
 # рекурентних мір  
  
RR = [] # Частота повторення  
DET = [] # Детермінізм  
DIV = [] # Розбіжність  
AVG\_DIAG\_LINE = [] # Усереднена довжина діагональних ліній  
ENT\_DIAG = [] # Ентропія діагональних ліній  
LAM = [] # Ламінарність  
TT = [] # Час затримки  
ENT\_VERT = [] # Ентропія вертикальних ліній  
ENT\_WHITE\_VERT = [] # Ентропія білих вертикальних ліній  
AVG\_WVERT\_LINE = [] # Усереднена довжина білих вертикальних ліній  
VERT\_DIV = [] # Розбіжність вертикальних ліній  
RATIO\_DET\_REC = [] # Відношення детермінізму до частоти повторень  
RATIO\_LAM\_DET = [] # Відношення ламінарності до детермінізму  
WHITE\_VERT\_DIV = [] # Розбіжність білих вертикальних ліній  
DIAG\_RR = [] # Діагональна частота рекурентних значень

Для подальших розрахунків ми використовуватимемо метод complexity\_rqa() бібліотеки neuralkit2. Синтаксис даного методу виглядає наступним чином:

**complexity\_rqa(signal, dimension=3, delay=1, tolerance='sd', min\_linelength=2, method='python', show=False)**

**Параметри**

* **signal** (*Union[list, np.ndarray, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **delay** (*int*) — затримка в часі.
* **dimension** (*int*) — розмірність вкладень, .
* **tolerance** (*float*) — радіус багатовимірного околу в межах якого шукаються рекурентні траєкторії (часто позначається як ), відстань, на якій дві точки даних вважаються схожими. Якщо "sd" (за замовчуванням), буде встановлено значення .
* **min\_linelength** (*int*) — мінімальна довжина діагональних та вертикальних ліній За замовчування дорівнює 2.
* **method** (*str*) — Може бути "pyrqa" для виконання рекурентного аналізу, але із використанням бібліотеки PyRQA (потребує додаткового встановлення).
* **show** (*bool*) — візуалізувати рекурентну матрицю.

**Повертає**

* **rqa** (*DataFrame*) — результати процедури RQA.
* **info** (*dict*) — словник, що містить інформацію відносно параметрів, що використовувались для виконання RQA.

Тепер можемо приступити до віконної процедури:

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 fragm = time\_ser.iloc[i:i+window].copy() # відбираємо фрагмент  
  
 fragm = transformation(fragm, ret\_type) # виконуємо процедуру   
 # трансформації ряду  
   
 resultRQA, \_ = nk.complexity\_rqa(fragm,  
 delay=tau,  
 dimension=m,  
 tolerance=eps)  
   
 # Обчислення відношення ламінарності до детермінізму  
 resultRQA['LamiDet'] = resultRQA['Laminarity']/resultRQA['Determinism']  
  
 # Обчислення дивергенції чорних вертикальних ліній  
 resultRQA['VDiv'] = 1./resultRQA['VMax']  
  
 # Обчислення дивергенції білих вертикальних ліній  
 resultRQA['WVDiv'] = 1./resultRQA['WMax']  
  
 RR.append(resultRQA['RecurrenceRate'])  
 DET.append(resultRQA['Determinism'])  
 DIV.append(resultRQA['Divergence'])   
 AVG\_DIAG\_LINE.append(resultRQA['L'])  
 ENT\_DIAG.append(resultRQA['LEn'])  
 LAM.append(resultRQA['Laminarity'])   
 TT.append(resultRQA['TrappingTime'])   
 ENT\_VERT.append(resultRQA['VEn'])  
 ENT\_WHITE\_VERT.append(resultRQA['WEn'])  
 AVG\_WVERT\_LINE.append(resultRQA['W'])   
 VERT\_DIV.append(resultRQA['VDiv'])  
 WHITE\_VERT\_DIV.append(resultRQA['WVDiv'])  
 RATIO\_DET\_REC.append(resultRQA['DeteRec'])   
 RATIO\_LAM\_DET.append(resultRQA['LamiDet'])  
 DIAG\_RR.append(resultRQA['DiagRec'])

100%|██████████| 7054/7054 [02:39<00:00, 44.16it/s]

Зберігаємо отримані результати в текстових файлах:

name = f"RQA\_classic\_name={symbol}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}\_m={m}\_ \  
 tau={tau}\_eps={eps}.txt"  
  
np.savetxt("RR" + name, RR)  
np.savetxt("DIAG\_RR" + name, DIAG\_RR)  
np.savetxt("DET" + name, DET)  
np.savetxt("DIV" + name, DIV)  
np.savetxt("VERT\_DIV" + name, VERT\_DIV)  
np.savetxt("WHITE\_VERT\_DIV" + name, WHITE\_VERT\_DIV)  
np.savetxt("LAM" + name, LAM)  
np.savetxt("TT" + name, TT)  
np.savetxt("AVG\_DIAG\_LINE" + name, AVG\_DIAG\_LINE)  
np.savetxt("AVG\_WRITE\_VERT\_LINE" + name, AVG\_WVERT\_LINE)  
np.savetxt("ENT\_DIAG" + name, ENT\_DIAG)  
np.savetxt("ENT\_VERT" + name, ENT\_VERT)  
np.savetxt("ENT\_WHITE\_VERT" + name, ENT\_WHITE\_VERT)  
np.savetxt("RATIO\_DET\_REC" + name, RATIO\_DET\_REC)  
np.savetxt("RATIO\_LAM\_DET" + name, RATIO\_LAM\_DET)

### 4.2.2 Рекурентні міри

Тепер займемося побудовою та інтерпретацією отриманих результатів. Для візуалізації графіків визначимо наступну функцію:

def plot\_recurrence\_measure(measure, label, clr="magenta"):  
  
 fig, ax = plt.subplots()  
  
 ax2 = ax.twinx()  
  
 ax2.spines.right.set\_position(("axes", 1.03))  
  
 p1, = ax.plot(time\_ser.index[window:length:tstep],   
 time\_ser.values[window:length:tstep],   
 "b-", label=fr"{ylabel}")  
 p2, = ax2.plot(time\_ser.index[window:length:tstep],  
 measure,   
 color=clr,   
 label=fr'${label}$')  
  
 ax.set\_xlabel(xlabel)  
 ax.set\_ylabel(f"{ylabel}")  
  
 ax.yaxis.label.set\_color(p1.get\_color())  
 ax2.yaxis.label.set\_color(p2.get\_color())  
  
 tkw = dict(size=2, width=1.5)  
  
 ax.tick\_params(axis='x', \*\*tkw)  
 ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
 ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
  
  
 ax2.legend(handles=[p1, p2])  
  
 plt.savefig(label +  
 f" RQA\_classic\_name={symbol}\_window={window}\_step={tstep}\_ \  
 rettype={ret\_type}\_m={m}\_tau={tau}\_eps={eps}.jpg")  
   
 plt.show();

#### 4.2.2.1 Частота рекурентності (Recurrence rate)

Найпростішим показником є частота рекурентності, яка визначає щільністю рекурентних точок на рекурентній діаграмі, ігноруючи лінію ідентичності:

де — кількість точок на траєкторії фазового простору.

**Частота рекурентності** відповідає ймовірності того, що певний стан повториться.

plot\_recurrence\_measure(measure=RR, label='RR')

|  |
| --- |
| Рисунок 4.5: Динаміка індексу Доу-Джонса та частоти рекурентності |

Як ми можемо бачити з представленого рисунку, міра рекурентності зростає при крахових подіях, що вказує на зростання ступеня самоорганізації та злагодженості торгівельної активності трейдерів на цьому ринку.

#### 4.2.2.2 Діагональна частота рекурентності (Diagonal recurrence rate)

Даний підхід базується на діагональних рекурентних профілях часового ряду. Діагональний рекурентний профіль кількісно оцінює кількість рекурентних точок на різних лагах, подібно до функції автокореляцій. Для отримання діагонального профілю рекурентностей просто підраховується частка рекурентних точок на діагоналях, розташованих в нижньому правому або нижньому лівому куті рекурентної діаграми, і будується графік як функція відстані від головної діагоналі, тобто лагу.

По іншому можна сказати, що **діагональна частота рекурентності** фіксує величину автокореляції на різних лагах.

plot\_recurrence\_measure(measure=DIAG\_RR, label='DRR')

|  |
| --- |
| Рисунок 4.6: Динаміка індексу Доу-Джонса та діагональної частоти рекурентності |

З представленого рисунку видно, що діагональна частота рекурентності зростає у передкризові та кризові стани, що вказує на зростання величини автокореляції, що в свою чергу демонструє зростання ступеню самоорганізації у кризові та передкризові стани.

#### 4.2.2.3 Детермінізм (Determinism)

Наступним показником можна визначити частку рекурентних траєкторій, які формують діагональні лінії мінімальної довжини . Ця міра називається **детермінізмом** і пов’язана з передбачуваністю динамічної системи:

де — частотний розподіл довжин діагональних ліній (тобто підраховує кількість діагональних профілів довжини ).

|  |
| --- |
| Додаткова інформація по детермінізму |
| Детерміновані системи характеризуються значною варіацією діагональних ліній різної довжинию. Періодичні сигнали будуть характеризуватися довгими діагональними лініями, в той час як для хаотичних сигналів діагональні лінії будуть короткими. Для стохастичним систем діагональні лінії взагалі будуть відсутніми, за винятком випадкових закономріностей, що утворюватимуть дуже короткі діагональні лінії.  Білий шум, наприклад, мав би рекурентну діаграму з майже ізольованими рекурентними точками та дуже малих відсотком діагональних ліній, тоді як детермінований процес демонстрував би дуже малу кількість одиночних рекурентностей, але велику щільність довгих діагональних ліній. |

plot\_recurrence\_measure(measure=DET, label='DET')

|  |
| --- |
| Рисунок 4.7: Динаміка індексу Доу-Джонса та детермінізму |

Як ми можемо бачити з представленого рисунку, як правило, у передкризові та кризові стани показник детермінізму починає зростати, що свідчить і про зростання ступеня передбачуваності (впорядкованості) флуктуацій системи.

#### 4.2.2.4 Ламінарність (Laminarity)

Кількість рекурентних станів, які утворюють вертикальні лінії, можна кількісно визначити таким же чином. Ця міра називається **ламінарністю** і пов’язана з кількістю ламінарних фаз (незмінностей) у системі:

де — частотний розподіл довжин вертикальних ліній, які мають довжину принаймні .

|  |
| --- |
| Додаткова інформація по ламінарності |
| Ламінарність характеризує ймовірність системи перебувати в ламінарному (незмінному) стані. Зі збільшенням ізольованих рекурентних точок у системі, міра ламінарності спадатиме. |

plot\_recurrence\_measure(measure=LAM, label='LAM')

|  |
| --- |
| Рисунок 4.8: Динаміка індексу Доу-Джонса та ламінарності |

Можна бачити, що при кризових станах ступінь ламінарності зростає. Як ми могли бачити, зростає і щільність діагональних точок, і загалом зростає кількість рекурентних траєкторій у фазовому просторі. Це вказує на те, що фінасовий індекс Доу-Джонса “застрягає” у стані кризи. Кризи характеризуються трендостійкістю, персистентністю та детермінованістю своєї поведінки.

#### 4.2.2.5 Середня довжина діагональних ліній (The average diagonal lines length)

Також можна виміряти **середню довжину діагональних ліній**. Cередня довжина діагональних лінії визначається як

Загалом цей показник характеризує середній період часу при якому дві траєкторії фазового простору знаходяться в достатній близькості один до одного.

|  |
| --- |
| Додаткова інформація по середній довжині діагональних ліній |
| Середня довжина діагональних ліній визначає середній час при якому система залишається передбачуваною. |

plot\_recurrence\_measure(measure=AVG\_DIAG\_LINE, label='AVG L')

|  |
| --- |
| Рисунок 4.9: Динаміка індексу Доу-Джонса та середньої довжини діагональних ліній |

Як і до цього, ми можемо бачити, що середній час перебування Доу-Джонса у детермінованому стані зростає під час кризових явищ, що говорить зростання ступеня колективізації трейдерів на ринку.

#### 4.2.2.6 Час захоплення/затримки (Trapping time)

Усереднена довжина діагональної лінії пов’язана із часом передбачуваності динамічної системи та **часом затримки**. У даному випадку ми середню довжину вертикальних ліній:

|  |
| --- |
| Додаткова інформація по середній довжині вертикальних ліній |
| Середня довжина вертикальних ліній визначає середній час перебування системи в ламінарному стані. Тобто, вона відповідає середньому періоду часу при якому система “завмирає” у певному стані. Очевидно, що зростання цiєї величини характеризує дедалi бiльший час затримки дослiджуваної системи в певному станi. |

plot\_recurrence\_measure(measure=TT, label='TT')

|  |
| --- |
| Рисунок 4.10: Динаміка індексу Доу-Джонса та час затримки |

На представленому рисунку видно, що зростає в (перед-)кризові стани, що вказує на потребу системи перебувати ще більший час у стані кризи.

#### 4.2.2.7 Середня довжина білих вертикальних лінії (Average white vertical lines length)

**Середня довжина білих вертикальних ліній** може бути визначена як

де — це частотний розподіл білих вертикальних ліній довжиною , а відповідає найменшій довжині білих вертикальних ліній (найменшому періоду повернення до стану рекурентності).

|  |
| --- |
| Додаткова інформація по середній довжині білих вертикальних ліній |
| Представлену міру можна охарактеризувати як середній горизонт *непередбачуваності* системи. |

plot\_recurrence\_measure(measure=AVG\_WVERT\_LINE, label='WVL\_{mean}')

|  |
| --- |
| Рисунок 4.11: Динаміка індексу Доу-Джонса та середньої довжини білих вертикальних ліній |

Зростання середньої довжини бiлих вертикальних лiнiй демонструє, що кризовi подiї характеризуються не лише детермiнiзмом динамiки фондового ринку, але i несхожiстю даних подiй у порiвняннi з попереднiми станами.

#### 4.2.2.8 Ентропія діагональних ліній (Diagonal lines entropy)

Для відповідних діагональних сегментів можна розрахувати необхідну кількість інформації для опису всього розподілу цього типу ліній. Імовірність того, що діагональна лінія має точну довжину , можна оцінити за частотним розподілом із . Ентропія Шеннона цієї ймовірності виглядає наступним чином:

Даний показник відображає складність досліджуваної структури.

|  |
| --- |
| Додаткова інформація по ентропії діагональних ліній |
| Для некорельованого шуму чи осциляцiй ми тримали б мале значення цієї ентропiї. Мале значення даної ентропії вказувало би на те, що розподіл діагональних ліній представляється асиметричним: існувала б невеличка частка діагональних ліній конкретної довжини, що характеризувала би всю рекурентність досліджуваної системи. Зростання даної ентропії характеризувало би зростання симетричності розподілу довжин діагональних ліній. |

plot\_recurrence\_measure(measure=ENT\_DIAG, label='DLEn')

|  |
| --- |
| Рисунок 4.12: Динаміка індексу Доу-Джонса та ентропії діагональних ліній |

Видно, що ентропія діагональних ліній зростає під час кризових явищ, що вказує на зростання впливу детермінованих процесів із різним ступенем передбачуваності.

#### 4.2.2.9 Ентропія вертикальних ліній (Vertical lines entropy)

Ми можемо визначити Шеннонівську ентропію для розподілу вертикальних структур рекурентної діаграми. Імовірність того, що вертикальна лінія має точну довжину , можна оцінити за частотним розподілом із . Ентропія Шеннона цієї ймовірності визначається як

Ця міра, по аналогії до попередньої ентропії, також є мірою складності системи.

|  |
| --- |
| Додаткова інформація по ентропії вертикальних ліній |
| Для синусоїдального процесу ми би очікували мале значення даної ентропії, оскільки це простий періодичний процес. Для складного процесу з пам’ятю ми би очiкуємо високе значення цього типу рекурентної ентропiї. Це означати- ме, що ламiнарнiсть процесу характеризуються рiзноманiтними перiодами довгостроковості пам’яті системи. |

plot\_recurrence\_measure(measure=ENT\_VERT, label='VLEn')

|  |
| --- |
| Рисунок 4.13: Динаміка індексу Доу-Джонса та ентропії вертикальних ліній |

На даному рисунку видно, що ентропія вертикальних ліній починає зростати під час крахових явищ, що вказує на зростання ступеню ламінарності, тобто зростання рівномірності розподілу вертикальних ліній різноманітних довжин.

#### 4.2.2.10 Дивергенція (Divergence)

Показник може надати нам інформацію про максимальний ступінь передбачуваності досліджуваного періоду. Зворотнє значення **максимальної довжини діагональних ліній** або **дивергенція (розбіжність)** може вказати нам на швидкість та тривалість розбіжності досліджуваних траєкторій. Даний показник можна визначити як

Дана міра схожа на старший показник Ляпунова. Однак взаємозв’язок між цією мірою та позитивним максимальним показником Ляпунова набагато складніший (щоб обчислити показник Ляпунова з RP, необхідно враховувати весь розподіл частот діагональних ліній). Дивергенція може мати тенденцію позитивного максимального показника Ляпунова, але не більше.

|  |
| --- |
| Додаткова інформація по дивергенції |
| Чим вище значення дивергенції, тим швидше розбігаються траєкторії фазового простору. І навпаки, чим нижче значення дивергенції, тим ближче досліджувані траєкторії прилягають один до одного. |

plot\_recurrence\_measure(measure=DIV, label='DIV')

|  |
| --- |
| Рисунок 4.14: Динаміка індексу Доу-Джонса та дивергенції |

Даний рисунок показує, що дивергенція діагональних ліній починає спадати в кризові та передкризові періоди, що також вказує на зростання ступеня впорядкованості динаміки системи в дані періоди часу.

#### 4.2.2.11 Дивергенція вертикальних ліній (Vertical line divergence)

Зворотнє значення **максимальної довжини вертикальних ліній** або **розбіжність вертикальних ліній** можна визначити як:

|  |
| --- |
| Додаткова інформація по дивергенції вертикальних ліній |
| Максимальна довижна вертикальних ліній надавала нам інформацію про максимальний ступінь незмінюваності системи. Вертикальна дивергенція дозволяє нам охарактеризувати швидкість настання або спаду ламінарності у системі. Чи вище значення , тим швидше система виходить із ламінарного стану. І навпаки, чим нижчий даний показник, тим ближче траєкторії фазового простору один до одного, і тим вищий ступінь ламінарності системи в конкретний момент часу. |

plot\_recurrence\_measure(measure=VERT\_DIV, label='VDIV')

|  |
| --- |
| Рисунок 4.15: Динаміка індексу Доу-Джонса та дивергенції вертикальних ліній |

На даному рисунку видно, що періоди криз характеризуються спадом вертикальної дивергенції, тобто зростанням кількості вертикальних структур, що характеризують ще більший ступінь ламінарності станів.

#### 4.2.2.12 Дивергенція білих вертикальних ліній

Зворотнє значення **максимальної довжини білих вертикальних ліній** () можна охарактеризувати як **дивергенцію білих вертикальних ліній**. Її можна визначити наступним чином:

Зростання даного показника має вказувати на зростання ступеня рекурентності системи, а його спад має демонструвати зростання непередбачуваності.

plot\_recurrence\_measure(measure=WHITE\_VERT\_DIV, label='WVDIV')

|  |
| --- |
| Рисунок 4.16: Динаміка індексу Доу-Джонса та дивергенції білих вертикальних ліній |

На даному рисунку видно, що дивергенція білих вертикальних ліній представляє доволі зашумлену динаміку, а тому не може бути використана в якості ефективного індикатора кризових явищ.

#### 4.2.2.13 Ентропія білих вертикальних ліній (White vertical lines entropy)

Імовірність того, що біла вертикальна лінія має точну довжину , можна оцінити за частотним розподілом із . Ентропія Шеннона цієї ймовірності,

де — мінімальна довжина білої вертикальної лінії.

plot\_recurrence\_measure(measure=ENT\_WHITE\_VERT, label='WVLEN')

|  |
| --- |
| Рисунок 4.17: Динаміка індексу Доу-Джонса та ентропії білих вертикальних ліній |

Видно, що ентропія білих вертикальних ліній спадає у кризові та передкризові періоди фондового ринку, що вказує на зростання загальної передбачуваності системи та зміщення розподілу білих вертикальних ліній до конкретних довжин. Тобто, їх розподіл у періоди криз стає менш симетричним, що вказує на поступове заміщення білих вертикальних ліній чорними.

#### 4.2.2.14 Співвідношення частоти рекурентності до детермінізму

Співвідношення між і () можна використовувати для виявлення прихованих фазових переходів у системи:

plot\_recurrence\_measure(measure=RATIO\_DET\_REC, label='RATIO\_1')

|  |
| --- |
| Рисунок 4.18: Динаміка індексу Доу-Джонса та співвідношення між мірою передбачуваності та рекурентності |

Даний показник спадає під час кризових явищ фондового ринку. Це говорить про те, що має зростати загальна щільність рекурентних точок, як ізольованих, так і просто розподілу вертикальних структур. Тобто, у кризові періоди представляється вищою за .

#### 4.2.2.15 Співвідношення ламінарності до детермінізму (LAM/DET)

Так само як і попередня міра, відношення ламінарності до детермінізму може дозволити нам виокремити приховані переходи в досліджуваному сигналі:

plot\_recurrence\_measure(measure=RATIO\_LAM\_DET, label='RATIO\_2')

|  |
| --- |
| Рисунок 4.19: Динаміка індексу Доу-Джонса та співвідношення між мірою ламінарності та детермінізмом |

Якщо виходити з динаміки показника , можна сказати, що загальний ступінь детермінізму починає переважати над ламінарністю під час кризових явищ.

Але по результатам представлених показників ми можемо сказати, що досліджувані крахові та передкрахові події характеризуються зростанням рекурентності, і подібного роду поведінка може бути використана в якості передвісника подальших криз.

# 5. Лабораторна робота № 4

**Тема.** Інформаційні методи оцінки складності.

**Мета.** Навчитися використовувати основні показники складності з теорії інформації для аналізу часових рядів.

## 5.1 Теоретичні відомості

### 5.1.1 Складність. Кількісні міри складності. Інформаційні методи оцінки складності.

Дане століття називають століттям складності. Сьогодні питання “що таке складність?” вивчають фізики, біологи, математики і інформатики, хоча при теперішніх досягненнях у розумінні оточуючого світу, однозначної відповіді на це питання немає.

З цієї причини, відповідно до ідеї І. Пригожина, будемо досліджувати прояви складності системи, застосовуючи при цьому сучасні методи кількісного аналізу складності.

Серед таких методів на увагу заслуговують: - інформаційно-ентропійні; - засновані на теорії хаосу; - скейлінгово-мультифрактальні.

Зрозуміло, виходячи з різної природи методів, покладених в основу формування міри складності, вони приділяють певні вимоги до часових рядів, що слугують вхідними даними. Наприклад, перші дві групи методів вимагають стаціонарності вхідних даних. При цьому мають різну чутливість до таких характеристик, як детермінованність, стохастичність, причинність та кореляції. Тому у подальшому, порівнюючи комплексно ефективність різних показників складності, на вказані обставини ми будемо звертати увагу, підкреслюючи спеціально застосовність того чи іншого показника для характеристики різних сторін складності досліджуваних систем.

Розгляд першої групи методів почнемо з добре відомої міри складності, запропонованої А. Колмогоровим.

**Колмогорівська складність**. Поняття колмогорівської складності (або, як ще говорять, алгоритмічної ентропії) з’явилося в 1960-і роки на стику теорії алгоритмів, теорії інформації і теорії ймовірностей.

Ідея А. Колмогорова полягала в тому, щоб вимірювати кількість інформації, що міститься в індивідуальних скінчених об’єктах (а не у випадкових величинах, як у шеннонівській теорії інформації). Виявилось, що це можливо (хоча лише з точністю до обмеженого доданку). А. Колмогоров запропонував вимірювати кількість інформації в скінчених об’єктах за допомогою теорії алгоритмів, визначивши складність об’єкту як мінімальну довжину програми, що породжує цей об’єкт. Дане визначення стало базисом алгоритмічної теорії інформації, а також алгоритмічної теорії ймовірностей: об’єкт вважається випадковим, якщо його складність наближена до максимальної.

Що ж собою являє колмогорівська складність і як її виміряти? На практиці ми часто стикаємося з програмами, які стискують файли (для економії місця в архіві). Найбільш поширені називаються zip, gzip, compress, rar, arj та інші. Застосувавши таку програму до деякого файлу (з текстом, даними, програмою), ми отримуємо його стислу версію (яка, як правило, коротше початкового файлу). За нею можна відновити початковий файл з допомогою парної програми-“декомпресора”. Отже, у першому наближенні колмогорівську складність файлу можна описати як довжину його стислої версії. Тим самим файл, що має регулярну структуру і добре стискуваний, має малу колмогорівську складність (порівняно з його довжиною). Навпаки, погано стискуваний файл має складність, близьку до довжини.

Припустимо, що ми маємо фіксований спосіб опису (декомпресор) . Для даного слова розглянемо всі його описи, тобто всі слова , для яких визначене рівне . Довжину найкоротшого з них і називають колмогорівською складністю слова при даному способі опису :

де позначає довжину слова . Індекс підкреслює, що визначення залежить від вибору способу .

Можна показати, що існують оптимальні способи опису. Спосіб опису тим краще, чим він коротше. Тому природно дати таке визначення: спосіб не гірше за спосіб , якщо при деякому і при всіх .

Отже, за Колмогоровим, складність об’єкту (наприклад, тексту — послідовності символів) — це довжина мінімальної програми яка виводить даний текст, а ентропія — це складність, що ділиться на довжину тексту. На жаль, це визначення чисто умоглядне. Надійного способу однозначно визначити цю програму не існує. Але є алгоритми, які фактично якраз і намагаються обчислити колмогорівські складність тексту і ентропію.

### 5.1.2 Оцінка складності Колмогорова за схемою Лемпела-Зіва

Універсальна (в сенсі застосовності до різних мовних систем) міра складності кінцевої символьної послідовності була запропонована Лемпелем і Зівом. У рамках їх підходу складність послідовності оцінюється числом кроків процесу, що її породжує. Припустимими (редакційними) операціями при цьому є:

1. генерація символу (необхідна, як мінімум, для синтезу елементів алфавіту) і
2. копіювання “готового” фрагмента з передісторії (тобто з уже синтезованої частини тексту).

Нехай — скінчений алфавіт, — текст (послідовність символів), складений з елементів ; — -й символ тексту; — фрагмент тексту з -го по -й символ включно ; — довжина тексту . Тоді схему синтезу послідовності можна представити у вигляді конкатенації

де — фрагмент , породжуваний на -му кроці, а $m=m\_{H}(S) — число кроків процесу. З усіляких схем породження обирається мінімальна за числом кроків. Таким чином, складність послідовності за Лемпелем-Зівом

Мінімальність числа кроків забезпечується вибором для копіювання на кожному кроці максимально довгого прототипу з передісторії. Якщо позначити через номер позиції, з якої починається копіювання на -му кроці, то довжина фрагмента копіювання

а сам -й компонент складнісного розкладання (1) можна записати у вигляді

Випадок відповідає ситуації, коли в позиції стоїть символ, який раніше не зустрічався. При цьому ми застосовуємо операцію генерації символу.

Будемо знаходити складність за Лемпелем-Зівом (LZ) для часового ряду, який являє собою, наприклад, щоденні значення індексу фондового ринку. Для дослідження динаміки LZ та порівняння з іншими фондовими ринками будемо знаходити дану міру складності для підряду фіксованої довжини (вікна). Для цього обчислимо логарифмічні прибутковості та перетворимо їх у послідовність бітів. При цьому можна задавати кількість станів, які диференційовані (система числення). Так, для двох різних станів маємо 0, 1, для трьох — 0, 1, 2 і т.д. Для двійкової системи кодування буде задаватися поріг по середньому значенню і стани, наприклад, прибутковостей (*ret*) кодуватимуться наступним чином:

Також можна визначити так звану пермутаційну складність Лемпеля-Зіва (PLZС). У даному випадку би будемо опиратись на процедуру реконструкції фазового простору, що згадувалась в лабораторних 2 і 3. Згідно пермутаційній процедурі ми будемо брати фрагмент ряду довжини , що слугує розмірностю реконструйованого атрактора, та замінюємо кожне значення ряду його порядковим індексом. На подальшому ресунку представлено часовий ряд та його можливі порядкові шаблони:

|  |
| --- |
| Рисунок 5.1: Фрагмент часового ряду (а) та 6 можливих порядкових шаблонів, що можуть бути в цьому сигналі (b) |

Алгоритм Лемпеля-Зіва виконує дві операції: (1) додає новий біт в уже існуючу послідовність; (2) копіює вже сформовану послідовність. Алгоритмічна складність представляє собою кількість таких операцій, необхідних для формування заданої послідовності.

Для випадкової послідовності довжини алгоритмічна складність обчислюється за виразом . Тоді відносна алгоритмічна складність знаходиться як відношення отриманої складності до складності випадкової послідовності: .

Однак навіть цього підходу може бути недостатньо. Справа в тому, що складні сигнали проявляють притаманну їм складність на різних просторових і часових масштабах, тобто мають масштабно інваріантні властивості. Вони, зокрема проявляються через степеневі закони розподілу. Тому розрахунки алгоритмічної складності на “поверховому” масштабі сигналу можуть бути неприйнятними і призводити до помилкових висновків.

Для подолання таких труднощів використовуються мультимасштабні методи, до розгляду яких ми і переходимо.

### 5.1.3 Процедура грануляції для мультискейлінгового дослідження часових рядів. Мультимасштабні міри складності

Ідея цієї групи методів включає дві послідовно виконувані процедури:

1. процес “грубого дроблення” (coarse graining — “грануляції”) початкового часового ряду — усереднення даних на сегментах, що не перетинаються, розмір яких (вікно усереднення) збільшуватиметься на одиницю при переході на наступний за величиною масштаб;
2. обчислення на кожному з масштабів певного (до сих пір мономасштабного) показника складності.

Процес “грубого дроблення” (“грануляція”) полягає в усереднені послідовних відліків ряду в межах вікон, що не перетинаються, а розмір яких — збільшується при переході від масштабу до масштабу. Кожен елемент “гранульованого” часового ряду знаходиться у відповідності до виразу:

де характеризує фактор масштабування. Довжина кожного “гранульованого” ряду залежить від розміру вікна рівна . Для масштабу рівного 1 “гранульований” ряд просто тотожний оригінальному.

|  |
| --- |
| Рисунок 5.2: Схематична ілюстрація процесу грубого дроблення (“грануляції”) початкового часового ряду для масштабів 2 і 3 |

Бібліотека neurokit2 представляє метод для обчислення як мономасштабного показника складності Лемпеля-Зіва, так і його мультимасштабного аналогу.

Синтаксис **мономасштабної** процедури виглядає наступним чином:

**complexity\_lempelziv(signal, delay=1, dimension=2, permutation=False, symbolize='mean', \*\*kwargs)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **delay** (*int*) — часова затримка, . Використовується лише тоді, коли permutation=True.
* **dimension** (*int*) — розмірність вкладень, . Використовується лише коли permutation=True.
* **permutation** (*bool*) — якщо значення True, поверне складність Лемпеля-Зіва на основі порядкових патернів.
* **symbolize** (*str*) — використовується тільки коли permutation=False. Метод перетворення неперервного сигналу на вході у символьний (дискретний) сигнал. За замовчуванням присвоює 0 та 1 значенням нижче та вище середнього. Може мати значення None, щоб пропустити процес (якщо вхідний сигнал вже є дискретним). Можна скористатися методом complexity\_symbolize() для використання іншої процедури символізації ряду.
* **kwargs** — інші аргументи, які передаються до complexity\_ordinalpatterns() (якщо permutation=True) або complexity\_symbolize().

**Повертає**

* **lzc** (*float*) — складність Лемпеля-Зіва (LZC).
* **info** (*dict*) — словник, що містить додаткову інформацію про параметри, які використовуються для обчислення LZC.

Синтаксис мультимасштабної процедури вже інший:

**entropy\_multiscale(signal, scale='default', dimension=3, tolerance='sd', method='MSEn', show=False, \*\*kwargs)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень або датафрейму.
* **scale** (*str* або *int* або *list*) — список масштабних коефіцієнтів, що використовуються для процедури крос-грануляції часового ряду. Якщо значення "default", буде використано range(len(signal) / (dimension + 10)). Якщо "max", використовуватиме всі масштаби до половини довжини сигналу. Якщо ціле число, створить діапазон до вказаного цілого числа.
* **dimension** (*int*) — розмірність вкладення, .
* **tolerance** (*float*) — поріг пропускання (часто позначається як ), відстань, на якій дві точки даних вважаються подібними. Якщо "sd" (за замовчуванням), буде встановлено значення .
* **method** (*str*) — яку версію мультимасштабного показника обчислювати. Переважна кількість показників за цим методом відповідають саме ентропійним підходам. Нас цікавитиме саме "LZC".
* **show** (*bool*) — візуалізувати залежність показника від масштабу.
* **kwargs** — необов’язкові аргументи.

**Повертає**

* *float* — точкова оцінка мультимасштабного показника окремого часового ряду, що відповідає площі під кривою значень цього показника, яка, по суті, є сумою вибіркових значень, наприклад, "LZC" в діапазоні масштабних коефіцієнтів.
* *dict* — словник, що містить додаткову інформацію про параметри, які використовуються для обчислення мультимасштабного показника. Значення показника, що відповідають кожному фактору "Scale", зберігаються під ключем "Value".

### 5.1.4 Шеннонівська складність

Ентропійний аналіз часових рядів за допомогою ентропійних показників різного роду буде проведено у наступних роботах. Зараз же ми розглянемо найпростішу з ентропій — ентропію Шеннона та порівняємо її можливості кількісно оцінювати складність часових послідовностей у порівнянні з мірою Лемпеля-Зіва.

Ентропія Шеннона — це статистичний квантифікатор, який широко використовується для характеристики складних процесів. Він здатний виявляти аспекти нелінійності в досліджуваних сигналах, сприяючи більш надійному поясненню нелінійної динаміки різних точок аналізу, що, в свою чергу, покращує розуміння природи складних систем, які характеризуються складністю та нерівноважністю. Окрім складності та нерівноважності, більшість, але не всі, складні системи також характеризуються неоднорідним розподілом зв’язків. Поняття ентропії було використано Шенноном в теорії інформації для передачі даних.

Ентропія - це міра невизначеності та випадковості в системі. Якщо припустити, що всі наявні дані належать до одного класу, то неважко передбачити клас нових даних. У цьому випадку ентропія дорівнює 0. Будучи величиною між 0 і 1, коли всі ймовірності рівні, ентропія набуває найбільшого значення. Невизначеність, що виникає, коли подія відбувається з ймовірністю , можна позначити як . Якщо ймовірність появи класу дорівнює 1, тоді ентропія мінімальна, . Відповідно до концепції Шеннона, якщо у нас наявні ймовірності реалізації певної події , на виході отримується кількість інформації, що необхідна для опису цієї події. Тоді, Шеннонівська ентропія може бути визначена як

Синтаксис методу для розрахунку Шеннонівської ентропії виглядає наступним чином:

**entropy\_shannon(signal=None, base=2, symbolize=None, show=False, freq=None, \*\*kwargs)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **base** (*float*) — основа логарифму, що за замовчуванням дорівнює 2, що дає одиницю в бітах. Зауважте, що scipy.stats.entropy() за замовчуванням використовує число Ейлера (np.e) (натуральний логарифм), що дає міру інформації, виражену в натах.
* **symbolize** (*str*) — метод приведення неперервного сигналу на вході у символьний (дискретний) сигнал. За замовчуванням дорівнює нулю, що пропускає процес (і припускає, що вхідні дані вже є дискретними).
* **show** (*bool*) — якщо значення True, виводить дискретність сигналу.
* **freq** (*np.array*) — замість сигналу можна надати вектор ймовірностей.
* **kwargs** — необов’язкові аргументи. Наразі не використовуються.

**Повертає**

* **shanen** (*float*) — Шеннонівську ентропію.
* **info** (*dict*) — словник, що містить додаткову інформацію про параметри, які використовуються для обчислення Шеннонівської ентопії.

### 5.1.5 Інформація Фішера

Інформацію Фішера було введено Р. А. Фішером у 1925 році як міру “внутрішньої точності” в теорії статистичних оцінок. Вона є центральною для багатьох статистичних галузей, що виходять далеко за межі теорії складності. Даний показник вимірює кількість інформації, яку спостережувана випадкова величина несе про невідомий параметр. В аналізі складності вимірюється кількість інформації, яку система несе “про себе”. Він базується на розкладанні за сингулярним значенням реконструйованого фазового простору. Значення показника Фішера зазвичай антикорельоване з іншими показниками складності (чим більше інформації система приховує про себе, тим більш передбачуваною і, відповідно, менш складною вона є).

Інформацію Фішера можна визначити, використовуючи метод fisher\_information() бібліотеки neurokit2. Її синтаксис виглядає наступним чином:

**fisher\_information(signal, delay=1, dimension=2)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **delay** (*int*) — затримка в часі, .
* **dimension** (*int*) — розмірність векторів фазового простору, .

**Повертає**

* **fi** (*float*) — обчислена міра інформації Фішера.
* **info** (*dict*) — словник, що містить додаткову інформацію про параметри, які використовуються для обчислення інформації Фішера.

### 5.1.6 Складність та параметри Хьорта

Параметри Хьорта — це показники статистичних властивостей, які спочатку були введені Хьортом (Hjorth, 1970) для опису загальних характеристик сигналів електроенцифалограми у кількох кількісних термінах, але які можуть бути застосовані до будь-якого часового ряду. Параметрами є активність, рухливість і складність:

* Параметр **активності** () — це просто дисперсія сигналу, яка відповідає середній потужності сигналу (якщо його середнє значення дорівнює 0).
* Параметр **рухливості** () являє собою середню частоту або частку середньоквадратичного відхилення спектра потужності. Він визначається як квадратний корінь з дисперсії першої похідної сигналу, поділений на дисперсію сигналу.
* Параметр **складності** () дає оцінку смуги пропускання сигналу, яка вказує на схожість форми сигналу з чистою синусоїдою (для якої значення сходиться до 1). Іншими словами, це міра “надмірної деталізації” по відношенню до “найм’якшої” можливої форми кривої. Параметр “Складність” визначається як відношення рухливості першої похідної сигналу до рухливості самого сигналу.

де та представляють перші та другі похідні сигналу, відповідно.

|  |
| --- |
| Рисунок 5.3: Характеристичні зміни форми кривої, що ілюструє залежність кожного параметра |

Бібліотека neurokit2 представляє метод для отримання відповідних показників. Її синтаксис виглядає наступним чином:

**complexity\_hjorth(signal)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.

**Повертає**

* **hjorth** (*float*) — показник складності Хьорта.
* **info** (*dict*) — словник, що містить додаткові показники Хьорта, такі як "Mobility" та "Activity".

### 5.1.7 Час декореляції

Час декореляції (decorrelation time, DT) визначається як час (у відліках) першого перетину нуля функції автокореляції. Коротший час декореляції відповідає менш корельованому сигналу. Наприклад, зменшення часу декореляції в сигналах електроенцифалограми спостерігається перед нападами, що пов’язано зі зменшенням потужності низьких частот.

Бібліотека neurokit2 представляє функціонал для визначення часу декореляції, а саме метод complexity\_decorrelation(). Її синтаксис є наступним:

**complexity\_decorrelation(signal)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (часовий ряд) у вигляді вектора значень.

**Повертає**

* *float* — час декореляції.
* *dict* — словник, що містить додаткову інформацію про додаткові показники.

### 5.1.8 Відносна грубість (нерівність, шорсткість)

Відносна шорсткість — це відношення локальної дисперсії (автоковаріації з лагом 1) до глобальної дисперсії (автоковаріації з лагом 0), яке можна використовувати для класифікації різних “шумів”. Його також можна використовувати як індекс для перевірки застосовності фрактального аналізу (показники фрактальності будуть використовуватись у наступних роботах).

Синтаксис даного методу в бібліотеці neurokit2 виглядає наступним чином:

**complexity\_relativeroughness(signal, \*\*kwargs)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (часовий ряд) у вигляді вектора значень.
* **kwargs** (*optional*) — інші аргументи, що потребуються методу nk.signal\_autocor().

**Повертає**

* **rr** (*float*) — значення відносної грубості.
* **info** (*dict*) — словник, що містить інформацію відносно параметрів, що використовувались для обчислення показника грубості.

### 5.1.9 Взаємна інформація

Коли йдеться про виявлення зв’язків між змінними, ми часто використовуємо кореляцію Пірсона. Проблема полягає в тому, що цей показник знаходить лише *лінійні* зв’язки, що іноді може призвести до неправильної інтерпретації зв’язку між двома змінними. Тим не менш, інші статистичні методи вимірюють нелінійні зв’язки, такі як **взаємна інформація (mutual information, MI)**.

Взаємна інформація між двома випадковими величинами вимірює нелінійний зв’язок між ними. Крім того, вона показує, **скільки інформації можна отримати з випадкової величини**, спостерігаючи за іншою випадковою величиною.

Вона тісно пов’язана з поняттям ентропії. Тобто, зменшення невизначеності випадкової величини пов’язане з отриманням інформації з іншої випадкової величини. Отже, високе значення взаємної інформації вказує на велике зменшення невизначеності, тоді як низьке значення вказує на мале зменшення. Якщо взаємна інформація дорівнює нулю, це означає, що дві випадкові величини є незалежними.

Взаємну інформацію можна розрахувати наступним чином:

де та ймовірності спостереження окремо або , а ймовірність спостереження одночасно та .

Основна відмінність між кореляцією та взаємною інформацією полягає в тому, що **кореляція є мірою лінійної залежності**, тоді як взаємна інформація вимірює загальну залежність (включаючи **нелінійні** зв’язки). Тому взаємна інформація виявляє залежності, які не залежать тільки від **коваріації**. Таким чином, взаємна інформація дорівнює нулю, коли дві випадкові величини є строго незалежними.

Бібліотека neurokit2 представляє інструментарій для знаходження взаємної інформації між двома сигналами та . У даній роботі ми спробуємо віднайти взаємну інформацію як між двома часовими рядами, так і **авто-взаємну інформацію**, подібно до автокореляції.

Синтаксис потрібної нам процедури виглядає наступним чином:

**mutual\_information(x, y, method='varoquaux', bins='default', \*\*kwargs)**

**Параметри**

* **x** (*Union[list, np.array, pd.Series]*) — масив значень.
* **y** (*Union[list, np.array, pd.Series]*) — масив значень.
* **method** (*str*) — метод для обчислення взаємної інформації: "nolitsa", "varoquaux", "knn", "max".
* **bins** (*int*) — кількість бінів гістограми. Використовується лише для "nolitsa" та "varoquaux". Якщо "default", кількість бінів оцінюється згідно методики Hacine-Gharbi (2018).
* **kwargs** — додаткові ключові аргументи для обраного методу.

**Повертає**

* *float* — розрахована взаємна інформація.

Існують різноманітні підходи до розрахунку взаємної інформації:

* **nolitsa**: Класична взаємна інформація (трохи швидше, ніж метод "sklearn").
* **varoquaux**: Застосовує фільтр Гауса до об’єднаної гістограми. Величину згладжування можна налаштовувати за допомогою аргументу sigma (за замовчуванням sigma=1).
* **knn**: Непараметрична (тобто не заснована на біннінгу) оцінка за найближчими сусідами. Додаткові параметри включають k (за замовчуванням, k=3), кількість найближчих сусідів для використання.
* **max**: Максимальний коефіцієнт взаємної інформації, тобто є максимальним при певній комбінації кількості бінів.

Існує безліч різноманітних показників складності, що базуються на теорії інформації та інших парадигах, які ми ще представлятимемо в подальшому. Розглянемо ефективність використання зазначених показників у якості індикаторів або індикаторів-передвісників крахових подій.

## 5.2 Хід роботи

Спочатку імпортуємо необхідні модулі для подальшої роботи:

import matplotlib.pyplot as plt   
import numpy as np  
import neurokit2 as nk  
import yfinance as yf  
import pandas as pd  
import scienceplots  
from tqdm import tqdm  
  
%matplotlib inline

І виконаємо налаштування рисунків для виведення:

plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься  
 # для виведення рисунків  
  
params = {  
 'figure.figsize': (8, 6), # встановлюємо ширину та висоту рисунків за замовчуванням  
 'font.size': 22, # розмір фонтів рисунку  
 'lines.linewidth': 2, # товщина ліній  
 'axes.titlesize': 'small', # розмір титулки над рисунком  
 "font.family": "sans-serif", # сімейство стилів підписів   
 "font.serif": ["Times"], # стиль підпису  
 'savefig.dpi': 300 # якість збережених зображень  
}  
  
plt.rcParams.update(params) # оновлення стилю згідно налаштувань

Цього разу розглянему можливість побудови індикаторів-передвісників на прикладі фондового індексу S&P 500, але, окрім цього, додамо ще Біткоїн для розрахунку взаємної інформації між фондовим ринком та криптовалютним. Очевидно, що фондовий індекс S&P 500 мав би проіснувати довше за Біткоїн. До того ж, криптовалютний ринок працює безперервно на відміну від фондового, а тому треба буде об’єднати значення двох активів за тими датами що співпадають.

**Виконуємо зчитування фондового індексу**:

symbol\_1 = '^GSPC' # Символ першого індексу  
start\_1 = "2014-01-01" # Дата початку зчитування даних  
end\_1 = "2023-08-24" # Дата закінчення зчитування даних  
  
data\_1 = yf.download(symbol\_1, start\_1, end\_1) # вивантажуємо дані  
time\_ser\_1 = data\_1['Adj Close'].copy() # зберігаємо саме ціни закриття  
  
xlabel = 'time, days' # підпис по вісі Ох   
ylabel\_1 = symbol\_1 # підпис по вісі Оу

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

**Виконуємо зчитування криптовалютного індексу**:

symbol\_2 = 'BTC-USD' # Символ другого індексу  
start\_2 = "2014-01-01" # Дата початку зчитування даних  
end\_2 = "2023-08-24" # Дата закінчення зчитування даних  
  
data\_2 = yf.download(symbol\_2, start\_2, end\_2) # вивантажуємо дані  
time\_ser\_2 = data\_2['Adj Close'].copy() # зберігаємо саме ціни закриття  
  
xlabel = 'time, days' # підпис по вісі Ох   
ylabel\_2 = symbol\_2 # підпис по вісі Оу

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

Тепер створимо новий масив даних, що об’єднуватиме в собі значення S&P 500 та BTC по їх спільним датам:

# приводимо значення індексів до типу DataFrame, щоб мати змогу їх об'єднати   
# за допомогою бібліотеки pandas  
df\_time\_ser\_1 = pd.DataFrame(time\_ser\_1)   
df\_time\_ser\_2 = pd.DataFrame(time\_ser\_2)  
  
  
joined = df\_time\_ser\_1.merge(df\_time\_ser\_2, # об'єднуємо по датам тієї бази, що містить   
 on='Date', # більше дат  
 how='left')   
  
joined = joined.rename(columns={joined.columns[0]: symbol\_1, # переіменовуємо колонки по   
 joined.columns[1]: symbol\_2}) # змінним symbol\_1 та symbol\_2  
  
joined = joined.dropna() # видаляємо рядки, що містять нульові значення

Виводимо отриману базу:

joined

|  | ^GSPC | BTC-USD |
| --- | --- | --- |
| Date |  |  |
| 2014-09-17 | 2001.569946 | 457.334015 |
| 2014-09-18 | 2011.359985 | 424.440002 |
| 2014-09-19 | 2010.400024 | 394.795990 |
| 2014-09-22 | 1994.290039 | 402.152008 |
| 2014-09-23 | 1982.770020 | 435.790985 |
| ... | ... | ... |
| 2023-08-17 | 4370.359863 | 26664.550781 |
| 2023-08-18 | 4369.709961 | 26049.556641 |
| 2023-08-21 | 4399.770020 | 26124.140625 |
| 2023-08-22 | 4387.549805 | 26031.656250 |
| 2023-08-23 | 4436.009766 | 26431.640625 |

І візуалізуємо сам графік. Спочатку оголосимо функцію для попарної візуалізації рядів зі збереженням їх абсолютних значень:

def plot\_pair(x\_values, y\_values, x\_label, y\_label, file\_name, clr="magenta"):  
  
 fig, ax = plt.subplots()  
  
 ax2 = ax.twinx()  
  
 ax2.spines.right.set\_position(("axes", 1.03))  
  
 p1, = ax.plot(x\_values,   
 y\_values[0],   
 "b-", label=fr"{y\_label[0]}")  
 p2, = ax2.plot(x\_values,  
 y\_values[1],   
 color=clr,   
 label=fr'${y\_label[1]}$')  
  
 ax.set\_xlabel(x\_label)  
 ax.set\_ylabel(f"{y\_label[0]}")  
  
 ax.yaxis.label.set\_color(p1.get\_color())  
 ax2.yaxis.label.set\_color(p2.get\_color())  
  
 tkw = dict(size=2, width=1.5)  
  
 ax.tick\_params(axis='x', \*\*tkw)  
 ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
 ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
  
  
 ax2.legend(handles=[p1, p2])  
  
 plt.savefig(file\_name + ".jpg")  
   
 plt.show();

і тепер візуалізуємо отримані ряди:

values\_plot = joined.iloc[:,0].values, joined.iloc[:,1].values  
ylabels = ylabel\_1, ylabel\_2  
file\_name = f'joined {symbol\_1}\_{symbol\_2}'

plot\_pair(joined.index, values\_plot, xlabel, ylabels, file\_name)

|  |
| --- |
| Рисунок 5.4: Динаміка індексу S&P 500 та Біткоїна за досліджуваний період |

### 5.2.1 Розрахунок взаємної інформації

Розглянемо взаємну інформацію як індикатор нелінійної кореляції між двома фінансовими активами, і спробуємо сказати, чи є між ними “істинний” взаємозв’язок. Виконуватимемо розрахунки із використанням алгоритму руховому вікна. Також визначимо функцію transform() для нормалізації ряду.

def transformation(signal, ret\_type):  
  
 for\_rec = signal.copy()  
  
 if ret\_type == 1: # Зважаючи на вид ряду, виконуємо  
 # необхідні перетворення  
 pass  
 elif ret\_type == 2:  
 for\_rec = for\_rec.diff()  
 elif ret\_type == 3:  
 for\_rec = for\_rec.pct\_change()  
 elif ret\_type == 4:  
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 elif ret\_type == 5:   
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 for\_rec = for\_rec.abs()  
 elif ret\_type == 6:  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
  
 for\_rec = for\_rec.dropna().values  
  
 return for\_rec

ret\_type = 6 # вид ряду  
window = 100 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(joined.iloc[:,0].values) # довжина самого ряду  
  
MI = [] # масив для віконної взаємної інформації

Тепер приступимо до розрахунків:

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 # відбираємо фрагменти  
 fragm\_1 = joined[symbol\_1][i:i+window]   
 fragm\_2 = joined[symbol\_2][i:i+window]  
  
 # виконуємо процедуру трансформації ряду   
 fragm\_1 = transformation(fragm\_1, ret\_type)   
 fragm\_2 = transformation(fragm\_2, ret\_type)  
  
 # розраховуємо взаємну інформацію   
 mut\_inf = nk.mutual\_information(fragm\_1, fragm\_2)  
   
 # та додаємо результат до масиву значень  
 MI.append(mut\_inf)

100%|██████████| 2149/2149 [00:03<00:00, 706.05it/s]

Зберігаємо отриманий результат у текстовому файлі:

np.savetxt(f"mutual\_inf\_name1={symbol\_1}\_name2={symbol\_2}\_ \  
 window={window}\_step={tstep}\_rettype={ret\_type}.txt" , MI)

Візуалізуємо результат між відповідними показниками:

fig, ax = plt.subplots(figsize=(13,8))  
  
ax2 = ax.twinx()  
ax3 = ax.twinx()  
  
ax2.spines.right.set\_position(("axes", 1.03))  
ax3.spines.right.set\_position(("axes", 1.12))  
  
p1, = ax.plot(joined.index[window:length:tstep],   
 joined[symbol\_1][window:length:tstep].values,   
 "b-",   
 label=fr"{symbol\_1}")  
p2, = ax2.plot(joined.index[window:length:tstep],  
 joined[symbol\_2][window:length:tstep].values,  
 'red',   
 label=fr"{symbol\_2}")  
p3, = ax3.plot(joined.index[window:length:tstep],  
 MI,  
 'magenta',   
 label=r"$MI$")   
  
  
ax.set\_xlabel(xlabel)  
ax.set\_ylabel(f"{symbol\_1}")  
  
ax.yaxis.label.set\_color(p1.get\_color())  
ax2.yaxis.label.set\_color(p2.get\_color())  
ax3.yaxis.label.set\_color(p3.get\_color())  
  
tkw = dict(size=3, width=1.5)  
  
ax.tick\_params(axis='x', \*\*tkw)  
ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
ax3.tick\_params(axis='y', colors=p3.get\_color(), \*\*tkw)  
  
ax3.legend(handles=[p1, p2, p3])  
  
plt.savefig(f"mutual\_inf\_name1={symbol\_1}\_name2={symbol\_2}\_ \  
 window={window}\_step={tstep}\_rettype={ret\_type}.jpg")  
  
plt.show();

|  |
| --- |
| Рисунок 5.5: Динаміка індексу S&P 500, Біткоїна та взаємної інформації |

Як ми можемо бачити з представленого рисунку, на фондовому та криптовалютному ринках дійсно спостерігалися фази зростання взаємної інформації між ними. Найкраще це видно напередодні кризи 2018-го року, під час 2019, після коронавірусної пандемії та напередодні 2023 року. Для даного індикатора залишається простір для експериментів, що можуть вивести його на рівень достатньо потужного передвісника криз на фондовому ринку чи криптовалютному.

Як вже зазначалося, окрім обчислення взаємної інформації для двох пар часових сигналів, ми можемо обчислити *авто*взаємну інформація, тобто взаємну інформацію ряду самого із собою по різним часовим лагам, як це було пророблено для автокореляції. Недолік автокореляції полягає в тому, що вони визначає саме *лінійний* зв’язок теперішніх значень з попередніми. Автовзаємна інформація в свою чергу є показником *нелінійного* зв’язку теперішніх значень із попередніми.

Для обчислення автовзаємної інформації визначимо наступну функцію:

def automut(x, maxlag):  
 n = len(x) # визначаємо довжину сигналу  
 lags = np.arange(0, maxlag, dtype="int") # оголошуємо масив лагів від 0 до maxlag  
 mi = np.zeros(len(lags)) # оголошуємо масив під значення взаємної інформації  
 for i, lag in enumerate(lags): # проходимось по кожному лагу  
   
 # виконуємо зміщення на lag значень   
 y1 = x[:n-lag].copy()  
 y2 = x[lag:].copy()  
  
 # і розраховуємо взаємну інформацію між часовим рядом y1  
 # та його зміщенною на lag кроків копією   
 mi[i] = nk.mutual\_information(y1, y2, bins=100)  
  
 return mi

Виведемо залежність автовзаємної інформації від лагу для всього ряду S&P 500 та Біткоїна. Спочатку розрахуємо вихідні значення ряду, далі прибутковості і потім волатильності. Для кожного з відповідних сигналів виведемо взаємну інформацію.

**Виконуємо перетворення S&P 500 та Біткоїна**

sp\_init = transformation(time\_ser\_1, ret\_type=1)  
sp\_ret = transformation(time\_ser\_1, ret\_type=4)  
sp\_vol = np.abs(sp\_ret.copy())  
  
btc\_init = transformation(time\_ser\_2, ret\_type=1)  
btc\_ret = transformation(time\_ser\_2, ret\_type=4)  
btc\_vol = np.abs(btc\_ret.copy())

**Розраховуємо автовзаємну інформацію S&P 500 та Біткоїна**

max\_lag = 100  
  
mu\_sp\_init = automut(sp\_init, max\_lag)  
mu\_sp\_ret = automut(sp\_ret, max\_lag)  
mu\_sp\_vol = automut(sp\_vol, max\_lag)  
  
mu\_btc\_init = automut(btc\_init, max\_lag)  
mu\_btc\_ret = automut(btc\_ret, max\_lag)  
mu\_btc\_vol = automut(btc\_vol, max\_lag)  
  
lags = np.arange(0, max\_lag, dtype="int") # оголошуємо масив лагів від 0 до maxlag

fig, ax = plt.subplots() # Створюємо порожній графік  
  
ax.plot(lags, mu\_sp\_init, label=r'$MI $ ' + f'{symbol\_1}') # Додаємо дані до графіку  
ax.plot(lags, mu\_sp\_ret, label=r'$MI$ ' + r'$g(t)$')   
ax.plot(lags, mu\_sp\_vol, label=r'$MI$ ' + r'$V\_{T}$')   
  
ax.legend() # Додаємо легенду  
ax.set\_xlabel("Lag") # Додаємо підпис для вісі Ох  
ax.set\_ylabel("Automutual information") # Додаємо підпис для вісі Оу  
  
plt.savefig(f'Automutual information {symbol\_1}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 5.6: Зміна з часом автовзаємної інформації для вихідного ряду x, нормалізованих прибутковостей g та модулів mod(g) фондового індексу S&P 500 |

fig, ax = plt.subplots() # Створюємо порожній графік  
  
ax.plot(lags, mu\_btc\_init, label=r'$MI $ ' + f'{symbol\_2}') # Додаємо дані до графіку  
ax.plot(lags, mu\_btc\_ret, label=r'$MI$ ' + r'$g(t)$')   
ax.plot(lags, mu\_btc\_vol, label=r'$MI$ ' + r'$V\_{T}$')   
  
ax.legend() # Додаємо легенду  
ax.set\_xlabel("Lag") # Додаємо підпис для вісі Ох  
ax.set\_ylabel("Automutual information") # Додаємо підпис для вісі Оу  
  
plt.savefig(f'Automutual information {symbol\_2}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 5.7: Зміна з часом автовзаємної інформації для вихідного ряду x, нормалізованих прибутковостей g та модулів mod(g) криптовалютного індексу BTC |

Як ми можемо бачити з представлених графіків, ступінь взаємної інформації це показник, що найкращим чином працює саме для вихідних значень часових сигналів. Для вихідного ряду ступінь взаємної інформації залишається доволі високим. Для прибутковостей і волатильностей взаємна інформація спадає одразу на першому лагу, що свідчить про незалежність значень на подальших часових затримках.

### 5.2.2 Розрахунок мономасштабної складності Лемпеля-Зіва

Продовжимо розраховувати й інші показники складності. Розглянемо можливість використання показника складності Лемпеля-Зіва в якості індикатора катастрофічних подій.

ret\_type = 4 # вид ряду  
window = 250 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(time\_ser\_1.values) # довжина самого ряду  
m = 4 # розмірність вкладень   
tau = 1 # часова затримка   
  
LZC = [] # класична складність Лемпеля-Зіва  
PLZC = [] # пермутаційна складність Лемпеля-Зіва

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 # відбираємо фрагмент  
 fragm = time\_ser\_1.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо класичну складність Лемпеля-Зіва   
 lzc, \_ = nk.complexity\_lempelziv(fragm)  
  
 # та пермутаційну складність Лемпеля-Зіва  
 plzc, \_ = nk.complexity\_lempelziv(fragm,   
 delay=tau,   
 dimension=m,   
 permutation=True)  
  
  
 # та додаємо результати до масиву значень  
 LZC.append(lzc)  
 PLZC.append(plzc)

100%|██████████| 2177/2177 [00:18<00:00, 116.70it/s]

Зберігаємо результати в текстових файлах:

np.savetxt(f"lzc\_name={symbol\_1}\_window={window}\_step={tstep}\_rettype={ret\_type}.txt" , LZC)  
np.savetxt(f"plzc\_name={symbol\_1}\_window={window}\_step={tstep}\_ \  
 rettype={ret\_type}\_m={m}\_tau={tau}.txt" , PLZC)

Та візуалізуємо їх:

fig, ax = plt.subplots(figsize=(13,8))  
  
ax2 = ax.twinx()  
ax3 = ax.twinx()  
  
ax2.spines.right.set\_position(("axes", 1.03))  
ax3.spines.right.set\_position(("axes", 1.12))  
  
p1, = ax.plot(time\_ser\_1.index[window:length:tstep],   
 time\_ser\_1.values[window:length:tstep],   
 "b-",   
 label=fr"{symbol\_1}")  
p2, = ax2.plot(time\_ser\_1.index[window:length:tstep],  
 LZC,  
 'gold',   
 label=fr"$LZC$")  
p3, = ax3.plot(time\_ser\_1.index[window:length:tstep],  
 PLZC,  
 'red',   
 label=fr"$PLZC$")   
  
  
ax.set\_xlabel(xlabel)  
ax.set\_ylabel(f"{symbol\_1}")  
  
ax.yaxis.label.set\_color(p1.get\_color())  
ax2.yaxis.label.set\_color(p2.get\_color())  
ax3.yaxis.label.set\_color(p3.get\_color())  
  
tkw = dict(size=3, width=1.5)  
  
ax.tick\_params(axis='x', \*\*tkw)  
ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
ax3.tick\_params(axis='y', colors=p3.get\_color(), \*\*tkw)  
  
ax3.legend(handles=[p1, p2, p3])  
  
plt.savefig(f"plzc\_lzc\_name={symbol\_1}\_ \  
 window={window}\_step={tstep}\_ \  
 rettype={ret\_type}\_m={m}\_tau={tau}.jpg")  
  
plt.show();

|  |
| --- |
| Рисунок 5.8: Динаміка індексу S&P 500, класичної мономасштабної складності Лемпеля-Зіва та її пермутаційного аналогу |

На даному рисунку видно, що 2 міри поводять себе асиметрично по відношенню один до одного: вказує на зростання складності, наприклад, події 2019 року. У той же час вказує на спад складності системи в цей період. Варто дослідити мультимасштабну динаміку міри Лемпеля-Зіва для більш змістовних висновків.

### 5.2.3 Обчислення мультимасштабної складності Лемпеля-Зіва

ret\_type = 4  
ret\_sp = transformation(time\_ser\_1, ret\_type)

mslzc, info = nk.entropy\_multiscale(ret\_sp, method="LZC",   
 scale=200, show=True)

|  |
| --- |
| Рисунок 5.9: Залежність від масштабу класичної складності Лемпеля-Зіва для S&P 500 |

**Мультимасштабна динаміка пермутаційного показника складності Лемпеля-Зіва**

msplzc, info = nk.entropy\_multiscale(ret\_sp,   
 method="LZC",   
 permutation=True,  
 dimension=m,  
 delay=tau,   
 scale=200,   
 show=True)

|  |
| --- |
| Рисунок 5.10: Залежність від масштабу пермутаційної складності Лемпеля-Зіва для S&P 500 |

Тепер розрахуємо віконну динаміку мультимасштабних показників Лемпеля-Зіва. Ми повертатимемо сумарну складність Лемпеля-Зіва за всіма масштабам.

ret\_type = 4 # вид ряду  
window = 250 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(time\_ser\_1.values) # довжина самого ряду  
m = 4 # розмірність вкладень   
tau = 1 # часова затримка   
  
MSLZC = [] # мультимасштабна складність Лемпеля-Зіва  
MSPLZC = [] # мультимасштабна пермутаційна складність Лемпеля-Зіва

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 # відбираємо фрагмент  
 fragm = time\_ser\_1.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо мультимасштабну складність Лемпеля-Зіва   
 mslzc, \_ = nk.entropy\_multiscale(fragm)  
  
 # та мультимасштабну пермутаційну складність Лемпеля-Зіва  
 msplzc, \_ = nk.entropy\_multiscale(fragm,   
 delay=tau,   
 dimension=m,   
 permutation=True)  
  
  
 # та додаємо результати до масиву значень  
 MSLZC.append(mslzc)  
 MSPLZC.append(msplzc)

100%|██████████| 2177/2177 [00:49<00:00, 43.90it/s]

np.savetxt(f"mslzc\_name={symbol\_1}\_window={window}\_step={tstep}\_ \  
 rettype={ret\_type}.txt" , MSLZC)  
np.savetxt(f"msplzc\_name={symbol\_1}\_window={window}\_step={tstep}\_ \  
 rettype={ret\_type}\_m={m}\_tau={tau}.txt" , MSPLZC)

fig, ax = plt.subplots(figsize=(13,8))  
  
ax2 = ax.twinx()  
ax3 = ax.twinx()  
  
ax2.spines.right.set\_position(("axes", 1.03))  
ax3.spines.right.set\_position(("axes", 1.12))  
  
p1, = ax.plot(time\_ser\_1.index[window:length:tstep],   
 time\_ser\_1.values[window:length:tstep],   
 "b-",   
 label=fr"{symbol\_1}")  
p2, = ax2.plot(time\_ser\_1.index[window:length:tstep],  
 MSLZC,  
 'gold',   
 label=fr"$MSLZC$")  
p3, = ax3.plot(time\_ser\_1.index[window:length:tstep],  
 MSPLZC,  
 'red',   
 label=fr"$MSPLZC$")   
  
  
ax.set\_xlabel(xlabel)  
ax.set\_ylabel(f"{symbol\_1}")  
  
ax.yaxis.label.set\_color(p1.get\_color())  
ax2.yaxis.label.set\_color(p2.get\_color())  
ax3.yaxis.label.set\_color(p3.get\_color())  
  
tkw = dict(size=3, width=1.5)  
  
ax.tick\_params(axis='x', \*\*tkw)  
ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
ax3.tick\_params(axis='y', colors=p3.get\_color(), \*\*tkw)  
  
ax3.legend(handles=[p1, p2, p3])  
  
plt.savefig(f"msplzc\_mslzc\_name={symbol\_1}\_ \  
 window={window}\_step={tstep}\_ \  
 rettype={ret\_type}\_m={m}\_tau={tau}.jpg")  
  
plt.show();

|  |
| --- |
| Рисунок 5.11: Динаміка індексу S&P 500, класичної мультимасштабної складності Лемпеля-Зіва та її пермутаційного аналогу |

Тепер бачимо однозначну картину: обидві міри поводять себе синхронно, та спадають у кризові та передкризові періоди, що вказує на зростання ступеня детермінованості та самоорганізації ринку.

### 5.2.4 Обчислення Шеннонівської ентропії

Як уже зазначалося, Шеннонівська ентропія — це міра непередбачуваності стану, або, еквівалентно, його середнього інформаційного вмісту. Ентропія Шеннона є однією з перших і найбільш базових мір ентропії та фундаментальним поняттям теорії інформації.

Розраховуватимемо її в ковзному вікні.

ret\_type = 1 # вид ряду  
window = 250 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(time\_ser\_1.values) # довжина самого ряду   
log\_base = np.exp(1)   
  
shannon = [] # ентропія Шеннона

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 # відбираємо фрагмент  
 fragm = time\_ser\_1.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо ентропію Шеннона  
 p, be = np.histogram(fragm, # розраховуємо щільність ймовірностей  
 bins='auto',   
 density=True)   
 r = be[1:] - be[:-1] # знаходимо dx  
 P = p \* r # представляємо ймовірність як f(x)\*dx  
 P = P[P!=0] # фільтруємо по всім ненульовим ймовірностям  
  
 sh\_ent, \_ = nk.entropy\_shannon(freq=P, base=log\_base) # розраховуємо ентропію   
 sh\_ent /= np.log(len(P)) # та нормалізуємо  
  
 # та додаємо результат до масиву значень  
 shannon.append(sh\_ent)

100%|██████████| 2177/2177 [00:01<00:00, 2135.92it/s]

np.savetxt(f"shannon\_ent\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}.txt" , shannon)

values\_plot = time\_ser\_1.values[window:length:tstep], shannon  
ylabels = ylabel\_1, "ShEn"  
file\_name = f"shannon\_ent\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}"

plot\_pair(time\_ser\_1.index[window:length:tstep],   
 values\_plot, xlabel, ylabels, file\_name)

|  |
| --- |
| Рисунок 5.12: Динаміка індексу S&P 500 та ентропії Шеннона |

Як ми можемо бачити з представленого рисунку, ентропія Шеннона реагує спадом на кризові періоди індексу S&P 500, що вказує на приріст ступеня періодизації системи, її детермінованості.

### 5.2.5 Розрахунок інформаційного показника Фішера

Перш за все задаємо параметри для розрахунків:

ret\_type = 1 # вид ряду  
window = 250 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(time\_ser\_1.values) # довжина самого ряду   
m = 3 # розмірність вкладень  
tau = 1 # часова затримка  
  
fisher = [] # інформація Фішера

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 # відбираємо фрагмент  
 fragm = time\_ser\_1.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 fish\_inf, \_ = nk.fisher\_information(signal=fragm,  
 dimension=m,   
 delay=tau)   
  
 # та додаємо результат до масиву значень  
 fisher.append(fish\_inf)

100%|██████████| 2177/2177 [00:00<00:00, 2937.25it/s]

np.savetxt(f"fisher\_inf\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}\_dimension={m}\_delay={tau}.txt", fisher)

values\_plot = time\_ser\_1.values[window:length:tstep], fisher  
ylabels = ylabel\_1, "FI"  
file\_name = f"fisher\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}\_dimension={m}\_delay={tau}"

plot\_pair(time\_ser\_1.index[window:length:tstep], values\_plot, xlabel, ylabels, file\_name)

|  |
| --- |
| Рисунок 5.13: Динаміка індексу S&P 500 та інформаційного показника Фішера |

На даному рисунку видно, що показник Фішера спадає у кризові та передкризові періоди, що говорить про спад кількості інформації, що необхідна для опису самоорганізованої динаміки фінансових криз, тобто зростання корельованості між діями трейдерів на ринку.

### 5.2.6 Обчислення часу декореляції

ret\_type = 1 # вид ряду  
window = 250 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(time\_ser\_1.values) # довжина самого ряду   
  
decorrelation\_time = [] # час декореляції

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 # відбираємо фрагмент  
 fragm = time\_ser\_1.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 dec\_time, \_ = nk.complexity\_decorrelation(fragm)   
  
 # та додаємо результат до масиву значень  
 decorrelation\_time.append(dec\_time)

100%|██████████| 2177/2177 [00:01<00:00, 1857.09it/s]

np.savetxt(f"dec\_time\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}.txt", decorrelation\_time)

values\_plot = time\_ser\_1.values[window:length:tstep], decorrelation\_time  
ylabels = ylabel\_1, "DT"  
file\_name = f"dec\_time\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}"

plot\_pair(time\_ser\_1.index[window:length:tstep], values\_plot,   
 xlabel, ylabels, file\_name)

|  |
| --- |
| Рисунок 5.14: Динаміка індексу S&P 500 та часу декореляції |

На представленому рисунку видно, що час декореляції зростає у період краху, що вказує на зростання кореляції системи в цей період.

### 5.2.7 Обчислення відносної шорсткості

ret\_type = 1 # вид ряду  
window = 250 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(time\_ser\_1.values) # довжина самого ряду   
  
relative\_roughness = [] # відносна шорсткість

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 # відбираємо фрагмент  
 fragm = time\_ser\_1.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 rr, \_ = nk.complexity\_relativeroughness(fragm)   
  
 # та додаємо результат до масиву значень  
 relative\_roughness.append(rr)

100%|██████████| 2177/2177 [00:01<00:00, 2094.81it/s]

np.savetxt(f"rel\_rough\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}.txt", relative\_roughness)

values\_plot = time\_ser\_1.values[window:length:tstep], relative\_roughness  
ylabels = ylabel\_1, "RR"  
file\_name = f"rel\_rough={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}"

plot\_pair(time\_ser\_1.index[window:length:tstep], values\_plot,   
 xlabel, ylabels, file\_name)

|  |
| --- |
| Рисунок 5.15: Динаміка індексу S&P 500 та показника відносної шорсткості |

Показник відносної шорсткості демонструє, що крахові події як, наприклад, у 2015, 2016, 2019, 2020 та 2023 роках характеризуються зростанням шорсткості своєї динаміка. Подібного роду поведінка є індикатором зростання шумової активності ринку: кореляційних характеристик та загальної варіації ринку в цілому. Зростання цього показника в періоди криз є індикатором зростання фрактальності ринку в дані періоди часу.

### 5.2.8 Розрахунок показників складності Хьорта

Завершуємо хід роботи показниками складності Хьорта:

ret\_type = 1 # вид ряду  
window = 250 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(time\_ser\_1.values) # довжина самого ряду   
  
activity = [] # параметр активності  
mobility = [] # параметр рухливості  
complexity = [] # параметр складності

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 # відбираємо фрагмент  
 fragm = time\_ser\_1.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо показники складності Хьорта  
 cmpl, info = nk.complexity\_hjorth(fragm)   
  
 # та додаємо результат до масиву значень  
 activity.append(info['Activity'])  
 mobility.append(info['Mobility'])  
 complexity.append(cmpl)

100%|██████████| 2177/2177 [00:00<00:00, 3338.20it/s]

np.savetxt(f"activity\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}.txt", activity)  
np.savetxt(f"mobility\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}.txt", mobility)  
np.savetxt(f"complexity\_name={symbol\_1}\_window={window}\_ \  
 step={tstep}\_rettype={ret\_type}.txt", complexity)

fig, ax = plt.subplots(figsize=(15,8))  
  
ax2 = ax.twinx()  
ax3 = ax.twinx()  
ax4 = ax.twinx()  
  
ax2.spines.right.set\_position(("axes", 1.03))  
ax3.spines.right.set\_position(("axes", 1.12))  
ax4.spines.right.set\_position(("axes", 1.19))  
  
p1, = ax.plot(time\_ser\_1.index[window:length:tstep],   
 time\_ser\_1.values[window:length:tstep],   
 "b-", label=fr"{ylabel\_1}")  
p2, = ax2.plot(time\_ser\_1.index[window:length:tstep],   
 activity, "r--", label=r"$Act$")  
p3, = ax3.plot(time\_ser\_1.index[window:length:tstep],   
 mobility, "g-", label=r"$Mob$")  
p4, = ax4.plot(time\_ser\_1.index[window:length:tstep],  
 complexity, "m-", label=r"$Comp$")  
  
ax.set\_xlabel(xlabel)  
ax.set\_ylabel(f"{ylabel\_1}")  
  
ax.yaxis.label.set\_color(p1.get\_color())  
ax2.yaxis.label.set\_color(p2.get\_color())  
ax3.yaxis.label.set\_color(p3.get\_color())  
ax4.yaxis.label.set\_color(p4.get\_color())  
  
tkw = dict(size=4, width=1.5)  
ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
ax3.tick\_params(axis='y', colors=p3.get\_color(), \*\*tkw)  
ax4.tick\_params(axis='y', colors=p4.get\_color(), \*\*tkw)  
ax.tick\_params(axis='x', \*\*tkw)  
  
ax4.legend(handles=[p1, p2, p3, p4])  
  
plt.savefig(f"hjorth\_name={symbol\_1}\_ret={ret\_type}\_wind={window}\_step={tstep}.jpg")  
plt.show();

|  |
| --- |
| Рисунок 5.16: Динаміка індексу S&P500 наряду з показниками активності, мобільності та складності Хьорта |

На даному рисунку видно, що параметр активності () представляється найменш інформативним, оскільки він вказує тільки на зростання сукупної дисперсії сигналу. Видно тільки те, що активність значно почала зростати напередодні 2022 року, але для попередніх кризових станів ми не бачимо передвісницької поведінки цього індикатора, тому він ще вимагатиме додактових досліджень та експериментів, що виходять за рамки даного посібника.

Питання передчасної ідентифікації наростання кризового явища найкраще вирішує показник мобільності (). Ми бачимо, що даний показник зростає під час 2015-2016 років, напередодні 2019, при настанні коронавірусної пандемії, перед 2023 роком та 2024.

Показник складності Хьорта () реагує асиметричним чином: у той час коли мобільність зростає, показник складності спадає, вказуючи на те, що динаміки системи прагне до вищого ступеня періодичності або корельованості.

# 6. Лабораторна робота № 5

**Тема.** Ентропійний аналіз складних систем.

**Мета.** Навчитись розраховувати значення різних типів ентропії часового ряду та досліджувати динаміку зміни її значень для оцінки якості прогнозів часових рядів.

## 6.1 Теоретичні відомості

Питання динаміки розвитку і функціонування складних систем може розглядатись у двох варіантах:

* як дослідження шумової активності;
* як детерміністичного випадку з певним ступенем порядку.

Останніми роками було використано кілька підходів для ідентифікації механізмів, що лежать в основі розвитку та функціонування складних систем. Особливо корисні результати було отримано при їх дослідженні методами теорії випадкових матриць, моно- та мультифрактального аналізу, теорії хаосу з реконструкцією траєкторії системи у фазовому просторі та визначення її параметрів, рекурентного аналузу. Ми розглянули ці методи у попередніх роботах. Однак, застосування деяких із методів висуває вимоги до стаціонарності досліджуваних даних, потребує довгих часових рядів та комплексного обчислення кількох параметрів.

Іншим підходом до розгляду питання вивчення особливостей складних систем є обчислення характеристик ентропії. Для практичного застосування у якості міри невизначеності, а значить і складності сигналу, використовують десятки різновидів ентропії.

Концепція термодинамічної ентропії як міри хаосу системи добре відома у фізиці, однак, останніми роками поняття ентропії було застосоване до складних систем інших об’єктів (біологічних, економічних, соціальних тощо). Так, один із найбільш часто використовуваних методів визначення ентропії базується на обчисленні спектру потужності Фур’є та застосовується для вивчення сигналів (часових рядів) різної природи. Проте, використання дискретного перетворення Фур’є для аналізу часових рядів має свої недоліки, зокрема, на результати впливає нестаціонарність рядів, варіювання їх довжини від сотень до сотень тисяч, та обмеження самого методу (незмінність частотно-часових характеристик протягом всього часу функціонування системи). Тому виникає питання про розрахунок значень ентропії за допомогою інших методів.

Введемо поняття ентропії, скориставшись інформацією, яку можна знайти у Вікіпедії.

**Термодинамічна ентропія** , часто просто іменована **ентропія**, в хімії і термодинаміці є мірою кількості енергії у фізичній системі, яка не може бути використана для виконання роботи. Вона також є мірою безладдя, присутнього в системі.

Поняття ентропії була вперше введено у 1865 році Рудольфом Клаузіусом. Він визначив зміну ентропії термодинамічної системи при оборотному процесі як відношення зміни загальної кількості тепла до величини абсолютної температури :

Рудольф Клаузіус дав величині ім’я “ентропія”, що походить від грецького слова τρoπή, “зміна” (зміна, перетворення). Зверніть увагу на те, що рівність відноситься до зміни ентропії.

У 1877 році, Людвіг Больцман зрозумів, що ентропія системи може відноситися до кількості можливих “мікростанів” (мікроскопічних станів) що узгоджуються з їх термодинамічними властивостями. Розглянемо, наприклад, ідеальний газ у посудині. Мікростан визначений як позиції і імпульси кожного атома, що становить систему. Зв’язність пред’являє до нас вимоги розглядати тільки ті мікростани, для яких: (i) місцерозташування всіх частин розташовані в рамках судини, (ii) для отримання загальної енергії газу кінетичні енергії атомів підсумовуються. Больцман постулював що

де константу ми знаємо тепер як сталу Больцмана, a є числом мікростанів, які можливі в наявному макроскопічному стані. Цей постулат, відомий як принцип Больцмана, може бути оцінений як початок статистичної механіки, яка описує термодинамічні системи використовуючи статистичну поведінку компонентів, із яких вони складаються. Принцип Больцмана зв’язує мікроскопічні властивості системи () з однією з її термодинамічних властивостей ().

Згідно визначенню Больцмана, ентропія є просто функцією стану. Більш того, оскільки () може бути тільки натуральним числом (1, 2, 3), ентропія повинна бути додатною — виходячи з властивостей логарифма.

У випадку дискретних станів квантової механіки кількість станів підраховується звичайним чином. В рамках класичної механіки мікроскопічний стан системи описується координатами й імпульсами окремих частинок, які пробігають неперервні значення. Для підрахунку станів у класичних системах фазовий простір розбивають на невеликі комірки із об’ємом, який відповідає сталій Планка. У такому випадку

де — число незалежних координат, — приведена стала Планка, а інтегрування проводиться по області фазового простору, який відповідає певному макроскопічному стану.

Клод Шеннон (Shannon, 1948) запропонував формулу для оцінки невизначеності кодової інформації в каналах зв’язку, звану ентропією Шеннона:

де — вірогідність того, що символ зустрічається в коді, який містить символів, — розмірний множник.

Зв’язок між ентропією і інформацією можна прослідкувати на наступному прикладі. Розглянемо тіло при абсолютному нулі температури, і хай ми маємо повну інформацію про координати і імпульси кожної частинки. Для простоти покладемо, що імпульси всіх частинок рівні нулю. В цьому випадку термодинамічна ймовірність рівна одиниці, а ентропія — нулю. При кінцевих температурах ентропія в рівновазі досягає максимуму. Можна зміряти всі макропараметри, що характеризують даний макростан. Проте ми практично нічого не знаємо про мікростан системи. Точніше кажучи, ми знаємо, що даний макростан можна реалізувати за допомогою дуже великого числа мікростанів. Таким чином, нульовій ентропії відповідає повна інформація (ступінь незнання рівний нулю), а максимальної ентропії — повне незнання мікростанів (ступінь незнання максимальний).

У теорії інформації ентропія (інформаційна ентропія) визначається як кількість інформації. Нехай — апріорна вірогідність деякої події (ймовірність до проведення досвіду), а – ймовірність цієї події після проведення досвіду. Для простоти вважатимемо, що . За Шенноном, кількість інформації , яка дає точну відповідь (після проведення експерименту)

Ця кількість інформації, за визначенням, дорівнює одному біту.

Фізичний сенс — це міра нашого незнання. Іншими словами, — це та інформація, яку ми можемо одержати, вирішивши завдання. У прикладі (тіло при абсолютному нулі температури), що розглядається вище, міра нашого незнання рівна нулю, оскільки . Після проведення досвіду ми одержуємо нульову інформацію , оскільки все було відомо до досвіду. Якщо розглядати тіло при кінцевих температурах, то до проведення досвіду число мікростанів, а отже, і дуже велике. Після проведення досвіду ми одержуємо велику інформацію, оскільки нам стають відомими координати і імпульси всіх частинок.

Аналогія між кількістю інформації і ентропією , визначуваною з принципу Больцмана, очевидна. Досить покласти множник рівним постійній Больцмана і використовувати натуральний логарифм. Саме з цієї причини величину $ I $ називають інформаційною ентропією. Інформаційна ентропія (кількість інформації) була визначена по аналогії із звичайною ентропією, і вона має властивості, характерні для звичайній ентропії: адитивність, екстремальні властивості і т.д. Проте ототожнювати звичайну ентропію з інформаційною не можна, оскільки неясно, яке відношення має друге начало до інформації. Нагадаємо, що екстенсивна величина — ця така характеристика системи, яка росте із збільшенням розмірів системи, тобто, якщо наша система складається з двох незалежних підсистем і , то ентропію всієї системи можна одержати складанням ентропій підсистем:

Саме ця властивість і означає екстенсивність, або адитивність, ентропії.

## 6.2 Хід роботи

import EntropyHub as eh  
import numpy as np  
from sklearn import preprocessing  
import matplotlib.pyplot as plt  
import pandas as pd  
import yfinance as yf  
#import antropy as ant  
import neurokit2  
import warnings  
warnings.filterwarnings('ignore')

plt.style.use('classic')  
  
params = {  
 'axes.labelsize': 26,  
 'axes.titlesize':26,   
 "axes.grid" : False,  
 'font.size': 26,   
 'legend.fontsize': 26,   
 'xtick.labelsize': 26,   
 'ytick.labelsize': 26,   
 'lines.linewidth': 2,  
 'axes.facecolor': 'white',  
 'figure.facecolor': 'white',  
 'axes.titlesize': 'small',  
 'font.family': 'Times New Roman',  
 'savefig.dpi': 300  
}  
  
plt.rcParams.update(params)  
  
xlabel = 'time, days'

symbol = "BTC-USD"  
symbol\_for\_graph = "BTC-USD"  
  
start = "2019-01-01"  
end = "2022-09-28"  
  
symbol\_plot = symbol.split('.')[0]  
symbol\_for\_graph\_plot = symbol\_for\_graph.split('.')[0]  
  
data = yf.download(symbol, start, end)  
close = data['Adj Close'].copy()  
for\_graph = data['Adj Close'].copy()  
  
np.savetxt(f'{symbol}\_initial\_time\_series.txt', close.values)

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

### 6.2.1 Виведення графіку досліджуваного ряду

fig, ax = plt.subplots()  
close.plot(figsize=(15,8), xlabel=xlabel, ylabel=symbol\_plot)  
ax.legend([symbol\_plot]);  
  
plt.savefig(f'{symbol}.jpg')  
plt.show()



fig, ax = plt.subplots()  
for\_graph.plot(figsize=(15,8), xlabel=xlabel, ylabel=symbol\_for\_graph\_plot)  
ax.legend([symbol\_for\_graph]);  
  
plt.savefig(f'{symbol\_for\_graph\_plot}.jpg')  
plt.show()



### 6.2.2 Задання ширини вікна та кроку

window = 100 # розмір вікна  
tstep = 1 # крок вікна

### 6.2.3 Approximate entropy (Апроксимаційна ентропія)

* Steven M. Pincus, Approximate entropy as a measure of system complexity, Proceedings of the National Academy of Sciences, 88.6 (1991): 2297-2301.

***Теоретичний опис методики розрахунку***

Ентропія подібності (Approximate Entropy, *ApEn*) є “статистикою регулярності”, що визначає можливість передбачувати флуктуації в часових рядах. Інтуїтивно вона означає, що наявність повторюваних шаблонів (послідовностей певної довжини, побудованих із чисел ряду, що слідують одне за іншим) флуктуацій у часовому ряді призводить до більшої передбачуваності часового ряду порівняно із рядами, де повторюваності шаблонів немає. Порівняно велике значення *ApEn* показує ймовірність того, що подібні між собою шаблони спостережень не будуть слідувати один за одним. Іншими словами, часовий ряд, що містить велику кількість повторюваних шаблонів, має порівняно мале значення *ApEn*, а значення *ApEn* для менш передбачуваного (більш складного) процесу є більшим.

При розрахунку *ApEn* для даного часового ряду , що складається із значень вибираються два параметри, та . Перший з цих параметрів, , вказує довжину шаблона, а другий — — визначає критерій подібності. Досліджуються підпослідовності елементів часового ряду , що складаються з чисел, взятих, починаючи з номера , і називаються векторами . Два вектори (шаблони), та $ p\_{m}(,j),$, будуть подібними, якщо всі різниці пар їх відповідних координат є меншими за значення , тобто якщо

Для розглядуваної множини всіх векторів довжини часового ряду можна обраховуються значення

де — кількість векторів у , що подібні вектору (враховуючи вибраний критерій подібності ). Значення є часткою векторів довжини , що мають схожість із вектором такої ж довжини, елементи якого починаються з номера . Для даного часового ряду обраховуються значення для кожного вектора у , після чого знаходиться середнє значення , яке виражає розповсюдженість подібних векторів довжини у ряду . Безпосередньо ентропія подібності для часового ряду з використанням векторів довжини та критерію подібності визначається за формулою:

тобто, як натуральний логарифм відношення повторюваності векторів довжиною до повторюваності векторів довжиною .

Таким чином, якщо знайдуться подібні вектори у часовому ряді, *ApEn* оцінить логарифмічну ймовірність того, що наступні інтервали після кожного із векторів будуть відрізнятись. Менші значення *ApEn* відповідають більшій ймовірності того, що за векторами слідують подібні їм. Якщо часовий ряд дуже нерегулярний — наявність подібних векторів не може бути передбачуваною і значення *ApEn* є порівняно великим.

Зауважимо, що *ApEn* є нестійкою до вхідних даних характеристикою, оскільки досить сильно залежить від параметрів та .

m = 3 #розмірність вкладень  
tau = 1 #часова затримка  
r = 0.45 #параметр подібності  
ret\_type = 4 #вид ряду: 1-вихідний, 2-абсолютні приб. 3-відносні приб. 4-нормалізовані приб.   
n = close.shape[0] #задаємо кількість значень  
  
ApEn = [] #масив для зберігання значень ентропії

for i in range(0,n-window,tstep):  
 fragm = close.iloc[i:i+window] #відбираємо фрагмент та в подальшому відбираємо потрібний тип ряду:  
   
 # 1 - вихідний ряд, 2 - детрендований, 3 - прибутковості, 4 - стандартизовані прибутковості, 5 - логарифмічні  
   
 if ret\_type == 1:  
 pass  
 elif ret\_type == 2:  
 fragm = fragm[1:] - fragm[:-1]  
 elif ret\_type == 3:  
 fragm = fragm.pct\_change()  
 elif ret\_type == 4:  
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
 elif ret\_type == 5:  
 fragm = np.log(fragm) - np.log(fragm.shift(1))  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
   
 fragm = fragm.dropna().values  
   
 Ap, \_ = eh.ApEn(Sig = fragm, m = m, tau = tau, r = r)  
 ApEn.append(Ap[-1])

name\_for\_save = f"ApEn\_{symbol}\_{window}\_{tstep}\_{m}\_{tau}\_{r}\_{ret\_type}.txt" #ім'я файлу для зберігання  
with open(name\_for\_save, 'w') as f: #відкриваємо на запис   
 ApEn\_file = [str(line) + '\n' for line in ApEn]  
 f.writelines(ApEn\_file)

fig, ax = plt.subplots(figsize=(15,8)) # візуалізуємо результати розрахунку  
  
ax.plot(close.index[window:n:tstep], for\_graph.values[window:n:tstep], label=f'{symbol\_for\_graph\_plot}', color='b')  
ax.set\_xlabel("time, days")  
ax.set\_ylabel(f"{symbol\_for\_graph\_plot}, ApEn")  
  
ax2 = ax.twinx()  
ax2.plot(close.index[window:n:tstep], ApEn, label='ApEn', color='r')  
  
fig.legend(loc="upper right", bbox\_to\_anchor=(1,1), bbox\_transform=ax.transAxes)  
  
plt.savefig(f"ApEn, symbol={symbol}, window size={window}, time step={tstep}, returns\_type={ret\_type}, embedding dimension={m}, delay={tau}, r={r}.jpg")  
plt.show()



### 6.2.4 Fuzzy entropy (Нечітка ентропія)

* Weiting Chen, et al. Characterization of surface EMG signal based on fuzzy entropy, IEEE Transactions on neural systems and rehabilitation engineering, 15.2 (2007): 266-272.
* Hong-Bo Xie, Wei-Xing He, and Hui Liu, Measuring time series regularity using nonlinear similarity-based sample entropy, Physics Letters A, 372.48 (2008): 7140-7146.

m = 3 #розмірність вкладень  
tau = 1 #часова затримка  
characteristic\_func = "default" #вид функції приналежності: default, sigmoid, gudermannian, linear  
r = (0.4, 2.0) #параметри, що подаються до функції приналежності. для default та sigmoid 2 значення r,   
 #для gudermannian та linear 1 значення r   
ret\_type = 4 #вид ряду: 1-вихідний, 2-абсолютні приб. 3-відносні приб. 4-нормалізовані приб.   
n = close.shape[0] #задаємо кількість значень  
  
FuzzEn = [] #масив для зберігання значень ентропії

for i in range(0,n-window,tstep):  
 fragm = close.iloc[i:i+window] #відбираємо фрагмент та в подальшому відбираємо потрібний тип ряду  
 if ret\_type == 1:   
 pass  
 elif ret\_type == 2:  
 fragm = fragm[1:] - fragm[:-1]  
 elif ret\_type == 3:  
 fragm = fragm.pct\_change()  
 elif ret\_type == 4:  
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
 elif ret\_type == 5:  
 fragm = np.log(fragm) - np.log(fragm.shift(1))  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
   
 fragm = fragm.dropna().values  
   
 Fuzz, \_, \_ = eh.FuzzEn(Sig = fragm, m = m, tau = tau, Fx = characteristic\_func, r = r) #Рахуємо нечітку ентропію   
 FuzzEn.append(Fuzz[-1]) #дожаємо розрахованє значення до масиву значень

name\_for\_save = f"FuzzEn\_{symbol}\_{window}\_{tstep}\_{m}\_{tau}\_{characteristic\_func}\_{r}\_{ret\_type}.txt" #ім'я файлу для зберігання  
with open(name\_for\_save, 'w') as f: #відкриваємо на запис   
 FuzzEn\_file = [str(line) + '\n' for line in FuzzEn]  
 f.writelines(FuzzEn\_file)

fig, ax = plt.subplots(figsize=(15,8))  
  
ax.plot(close.index[window:n:tstep], close.values[window:n:tstep], label=f"{symbol\_for\_graph\_plot}", color='b')  
ax.set\_xlabel("time, days")  
ax.set\_ylabel(f"{symbol\_for\_graph\_plot}, FuzzEn")  
  
ax2 = ax.twinx()  
ax2.plot(close.index[window:n:tstep], FuzzEn, label="FuzzEn", color='r')  
  
fig.legend(loc="upper right", bbox\_to\_anchor=(1,1), bbox\_transform=ax.transAxes)  
  
plt.savefig(f"FuzzEn, symbol={symbol}, window size={window}, time step={tstep}, returns\_type={ret\_type}, membership function={characteristic\_func}, embedding dimension={m}, delay={tau}, r={r}.jpg")  
  
plt.show()



### 6.2.5 Sample entropy (Ентропія шаблонів)

* Joshua S Richman and J. Randall Moorman, Physiological time-series analysis using approximate entropy and sample entropy, American Journal of Physiology-Heart and Circulatory Physiology (2000).

m = 3 #розмірність вкладень  
tau = 1 #часова затримка  
r = 0.4 #параметр подібності  
ret\_type = 4 #вид ряду: 1-вихідний, 2-абсолютні приб. 3-відносні приб. 4-нормалізовані приб.   
n = close.shape[0] #задаємо кількість значень  
  
SampEn = [] #масив для зберігання значень ентропії

for i in range(0,n-window,tstep):  
 fragm = close.iloc[i:i+window] #відбираємо фрагмент та в подальшому відбираємо потрібний тип ряду  
 if ret\_type == 1:  
 pass  
 elif ret\_type == 2:  
 fragm = fragm[1:] - fragm[:-1]  
 elif ret\_type == 3:  
 fragm = fragm.pct\_change()  
 elif ret\_type == 4:  
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
 elif ret\_type == 5:  
 fragm = np.log(fragm) - np.log(fragm.shift(1))  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
   
 fragm = fragm.dropna().values  
   
 Samp, \_, \_ = eh.SampEn(Sig = fragm, m = m, tau = tau, r = r, Logx=np.exp(1))  
 SampEn.append(Samp[-1])

name\_for\_save = f"SampEn\_{symbol}\_{window}\_{tstep}\_{m}\_{tau}\_{r}\_{ret\_type}.txt" #ім'я файлу для зберігання  
with open(name\_for\_save, 'w') as f: #відкриваємо на запис   
 SampEn\_file = [str(line) + '\n' for line in SampEn]  
 f.writelines(SampEn\_file)

fig, ax = plt.subplots(figsize=(15,8))  
  
ax.plot(close.index[window:n:tstep], close.values[window:n:tstep], label=f"{symbol\_for\_graph\_plot}", color='b')  
ax.set\_xlabel("time, days")  
ax.set\_ylabel(f"{symbol\_for\_graph\_plot}, SampEn")  
  
ax2 = ax.twinx()  
ax2.plot(close.index[window:n:tstep], SampEn, label="SampEn", color='r')  
  
fig.legend(loc="upper right", bbox\_to\_anchor=(1,1), bbox\_transform=ax.transAxes)  
  
plt.savefig(f"SampEn, symbol={symbol}, window size={window}, time step={tstep}, returns\_type={ret\_type}, embedding dimension={m}, delay={tau}, r={r}.jpg")  
  
plt.show()



### 6.2.6 Permutation entropy (Ентропія перестановок)

* Christoph Bandt and Bernd Pompe, Permutation entropy: A natural complexity measure for time series, Physical Review Letters, 88.17 (2002): 174102.
* Xiao-Feng Liu, and Wang Yue, Fine-grained permutation entropy as a measure of natural complexity for time series, Chinese Physics B, 18.7 (2009): 2690.
* Chunhua Bian, et al., Modiﬁed permutation-entropy analysis of heartbeat dynamics, Physical Review E, 85.2 (2012) : 021906
* Bilal Fadlallah, et al., Weighted-permutation entropy: A complexity measure for time series incorporating amplitude information, Physical Review E, 87.2 (2013): 022911.
* Hamed Azami and Javier Escudero, Amplitude-aware permutation entropy: Illustration in spike detection and signal segmentation, Computer methods and programs in biomedicine, 128 (2016): 40-51.
* Zhiqiang Huo, et al., Edge Permutation Entropy: An Improved Entropy Measure for Time-Series Analysis, 45th Annual Conference of the IEEE Industrial Electronics Soc, (2019), 5998-6003.
* Zhe Chen, et al., Improved permutation entropy for measuring complexity of time series under noisy condition, Complexity, 1403829 (2019).
* Maik Riedl, Andreas MuЁller, and Niels Wessel, Practical considerations of permutation entropy, The European Physical Journal Special Topics, 222.2 (2013): 249-262.

m = 4 # розмірність вкладень  
tau = 3 # часова затримка  
Type = 'weighted' # none - класична;   
 # finegrain - Fine-grained permutation entropy;   
 # modified - Modiﬁed permutation entropy;   
 # weighted - Weighted permutation entropy;   
 # ampaware - Amplitude-aware permutation entropy;   
 # edge - Edge permutation entropy;   
 # uniquant - Uniform quantization-based permutation entropy;   
   
tpx = -1 # finegrain tpx is the α parameter, a positive scalar (default: 1)  
 # ampaware tpx is the A parameter, a value in range [0 1] (default: 0.5)  
 # edge tpx is the r sensitivity parameter, a scalar > 0 (default: 1)  
 # uniquant tpx is the L parameter, an integer > 1 (default: 4).  
  
log = np.exp(1) # основа логарифма  
norm = True  
ret\_type = 4 # вид ряду: 1-вихідний, 2-абсолютні приб. 3-відносні приб. 4-нормалізовані приб.   
n = close.shape[0] # задаємо кількість значень  
  
PEn = [] # масив для зберігання значень нормалізованої перм. ентропії  
CPEn = [] # масив для зберігання значень умовної перм. ентропії

for i in range(0,n-window,tstep):  
 fragm = close.iloc[i:i+window].copy() # відбираємо фрагмент та в подальшому відбираємо потрібний тип ряду  
 if ret\_type == 1:  
 pass  
 elif ret\_type == 2:  
 fragm = fragm[1:] - fragm[:-1]  
 elif ret\_type == 3:  
 fragm = fragm.pct\_change()  
 elif ret\_type == 4:  
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
   
 fragm = fragm.dropna().values  
   
 \_, Pnorm, cPE = eh.PermEn(fragm, m = m, tau = tau, Typex = Type, tpx = tpx, Logx = log, Norm = norm)  
 PEn.append(Pnorm[-1])  
 CPEn.append(cPE[-1])

PEn\_for\_save = f"PEn\_symbol={symbol}\_window={window}\_step={tstep}\_d={m}\_tau={tau}\_ret={ret\_type}\_type={Type}\_param={tpx}.txt" #ім'я файлу для зберігання пермутаційної ентропії  
CPEn\_for\_save = f"CPEn\_symbol={symbol}\_window={window}\_step={tstep}\_d={m}\_tau={tau}\_ret={ret\_type}\_type={Type}\_param={tpx}.txt" #ім'я файлу для зберігання умовної пермутаційної ентропії  
with open(PEn\_for\_save, 'w') as f: #відкриваємо на запис   
 PEn\_file = [str(line) + '\n' for line in PEn]  
 f.writelines(PEn\_file)  
   
with open(CPEn\_for\_save, 'w') as f: #відкриваємо на запис   
 CPEn\_file = [str(line) + '\n' for line in CPEn]  
 f.writelines(CPEn\_file)

fig, ax = plt.subplots(figsize=(15,8))  
  
ax.plot(close.index[window:n:tstep], for\_graph.values[window:n:tstep], label=f"{symbol\_for\_graph\_plot}", color='b')  
ax.set\_xlabel("time, days")  
ax.set\_ylabel(f"{symbol\_for\_graph\_plot}, PEn")  
  
ax2 = ax.twinx()  
ax2.plot(close.index[window:n:tstep], PEn, label="PEn", color='r')  
  
fig.legend(loc="upper right", bbox\_to\_anchor=(1,1), bbox\_transform=ax.transAxes)  
  
plt.savefig(f"PEn, symbol={symbol}, window size={window}, time step={tstep}, returns\_type={ret\_type}, type={Type}, tpx={tpx}, embedding dimension={m}, delay={tau}.jpg")  
  
plt.show()



fig, ax = plt.subplots(figsize=(15,8))  
  
ax.plot(close.index[window:n:tstep], for\_graph.values[window:n:tstep], label=f"{symbol\_for\_graph\_plot}", color='b')  
ax.set\_xlabel("time, days")  
ax.set\_ylabel(f"{symbol\_for\_graph\_plot}, CPEn")  
  
ax2 = ax.twinx()  
ax2.plot(close.index[window:n:tstep], CPEn, label="CPEn", color='r')  
  
fig.legend(loc="upper right", bbox\_to\_anchor=(1,1), bbox\_transform=ax.transAxes)  
  
plt.savefig(f"CPEn, symbol={symbol}, window size={window}, time step={tstep}, returns\_type={ret\_type}, type={Type}, tpx={tpx}, embedding dimension={m}, delay={tau}.jpg")  
  
plt.show()



### 6.2.7 Distribution entropy (Розподільна ентропія)

* Li, Peng, et al., Assessing the complexity of short-term heartbeat interval series by distribution entropy, Medical & biological engineering & computing 53.1 (2015): 77-87.

m = 3 #розмірність вкладень  
tau = 1 #часова затримка  
bins = 'sturges' # Метод визначення бінів гістограми. Окрім цього можна обрати sqrt, rice, doanes  
norm = True  
log = np.exp(1)  
ret\_type = 4 #вид ряду: 1-вихідний, 2-абсолютні приб., 3-відносні приб., 4-нормалізовані приб.   
n = close.shape[0] #задаємо кількість значень  
  
DistEn = [] #масив значень для зберігання розподіленої ентропії

for i in range(0,n-window,tstep):  
 fragm = close.iloc[i:i+window] #відбираємо фрагмент та в подальшому відбираємо потрібний тип ряду  
 if ret\_type == 1:  
 pass  
 elif ret\_type == 2:  
 fragm = fragm[1:] - fragm[:-1]  
 elif ret\_type == 3:  
 fragm = fragm.pct\_change()  
 elif ret\_type == 4:  
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
   
 fragm = fragm.dropna().values  
   
 Dist = eh.DistEn(fragm, m = m, tau = tau, Bins = bins, Logx = log, Norm = norm)  
 DistEn.append(Dist[0])

Note: 3/14 bins were empty  
Note: 3/14 bins were empty  
Note: 3/14 bins were empty  
Note: 12/14 bins were empty  
Note: 12/14 bins were empty  
Note: 12/14 bins were empty  
Note: 9/14 bins were empty  
Note: 7/14 bins were empty  
Note: 4/14 bins were empty  
Note: 2/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty  
Note: 1/14 bins were empty

DistEn\_for\_save = f"DistEn\_symbol={symbol}\_window={window}\_step={tstep}\_d={m}\_tau={tau}\_series\_type={ret\_type}\_bins={bins}.txt" #ім'я файлу для зберігання розподіленої ентропії  
with open(DistEn\_for\_save, 'w') as f: #відкриваємо на запис   
 DistEn\_file = [str(line) + '\n' for line in DistEn]  
 f.writelines(DistEn\_file)

fig, ax = plt.subplots(figsize=(15,8))  
  
ax.plot(close.index[window:n:tstep], for\_graph.values[window:n:tstep], label=f"{symbol\_for\_graph\_plot}", color='b')  
ax.set\_xlabel("time, days")  
ax.set\_ylabel(f"{symbol\_for\_graph\_plot}, DistEn")  
  
ax2 = ax.twinx()  
ax2.plot(close.index[window:n:tstep], DistEn, label="DistEn", color='r')  
  
fig.legend(loc="upper right", bbox\_to\_anchor=(1,1), bbox\_transform=ax.transAxes)  
  
plt.savefig(f"DistEn\_symbol={symbol}\_window={window}\_step={tstep}\_d={m}\_tau={tau}\_series\_type={ret\_type}\_bins={bins}.jpg")  
  
plt.show()



### 6.2.8 Dispersion entropy (Дисперсійна ентропія)

* Mostafa Rostaghi and Hamed Azami, Dispersion entropy: A measure for time-series analysis IEEE Signal Processing Letters 23.5 (2016): 610-614.
* Hamed Azami and Javier Escudero, Amplitude-and ﬂuctuation-based dispersion entropy, Entropy 20.3 (2018): 210.
* Li Yuxing, Xiang Gao and Long Wang, Reverse dispersion entropy: A new complexity measure for sensor signal, Sensors 19.23 (2019): 5203.
* Wenlong Fu, et al., Fault diagnosis for rolling bearings based on ﬁne-sorted dispersion entropy and SVM optimized with mutation SCA-PSO, Entropy 21.4 (2019): 404.

***Теоретичний опис методики розрахунку***

norm = True  
fluct = False # Якщо True повертаємо флуктуаційно-дисперсійну ентропію  
m = 3   
tau = 1  
rho = 1 # \*If Typex = "finesort", rho is the tuning parameter, a positive scalar (default:1)  
  
classes = 3 # кількість символів, що задіяні при перетворені  
Type = 'ncdf' # тип символьного перетворення. "ncdf" Normalised cumulative distribution function [19]  
 # "kmeans" K-means clustering algorithm. Note: The ”kmeans” algorithm uses random initialization conditions. This causes results to vary slightly each time it is called.  
 # "linear" Linear segmentation of signal range  
 # "finesort" Fine-sorted dispersion entropy  
 # "equal" Approx. equal number of symbols.  
  
log = np.exp(1)  
ret\_type = 4 #вид ряду: 1-вихідний, 2-абсолютні приб., 3-відносні приб., 4-нормалізовані приб.   
n = close.shape[0] #задаємо кількість значень  
  
DispEn = [] # масив значень для зберігання дисперсійної ентропії   
RevDispEn = [] # для зберігання оборотної дисперсійної ентропії

for i in range(0,n-window,tstep):  
 fragm = close.iloc[i:i+window] #відбираємо фрагмент та в подальшому відбираємо потрібний тип ряду  
 if ret\_type == 1:  
 pass  
 elif ret\_type == 2:  
 fragm = fragm[1:] - fragm[:-1]  
 elif ret\_type == 3:  
 fragm = fragm.pct\_change()  
 elif ret\_type == 4:  
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
   
 fragm = fragm.dropna().values  
   
 Disp, RevDisp = eh.DispEn(fragm, m = m, tau = tau, c = classes, Typex = Type,   
 Logx = log, Fluct = fluct, Norm = norm, rho = rho)  
 DispEn.append(Disp)  
 RevDispEn.append(RevDisp)

DispEn\_for\_save = f"DispEn\_symbol={symbol}\_window={window}\_step={tstep}\_d\_e={m}\_tau={tau}\_series\_type={ret\_type}\_fluct={fluct}\_rho={rho}\_classes={classes}\_type={Type}.txt" #ім'я файлу для зберігання DispEn  
RevDispEn\_for\_save = f"RevDispEn\_symbol={symbol}\_window={window}\_step={tstep}\_d\_e={m}\_tau={tau}\_series\_type={ret\_type}\_fluct={fluct}\_rho={rho}\_classes={classes}\_type={Type}.txt" #ім'я файлу для зберігання RevDispEn  
with open(DispEn\_for\_save, 'w') as f: #відкриваємо на запис   
 DispEn\_file = [str(line) + '\n' for line in DispEn]  
 f.writelines(DispEn\_file)  
   
with open(RevDispEn\_for\_save, 'w') as f: #відкриваємо на запис   
 RevDispEn\_file = [str(line) + '\n' for line in RevDispEn]  
 f.writelines(RevDispEn\_file)

fig, ax = plt.subplots(figsize=(15,8))  
  
ax.plot(close.index[window:n:tstep], for\_graph.values[window:n:tstep], label=f"{symbol\_for\_graph\_plot}", color='b')  
ax.set\_xlabel("time, days")  
ax.set\_ylabel(f"{symbol\_for\_graph\_plot}, DispEn")  
  
ax2 = ax.twinx()  
ax2.plot(close.index[window:n:tstep], DispEn, label="DispEn", color='r')  
  
fig.legend(loc="upper right", bbox\_to\_anchor=(1,1), bbox\_transform=ax.transAxes)  
  
plt.savefig(f"DispEn\_symbol={symbol}\_window={window}\_step={tstep}\_d\_e={m}\_tau={tau}\_series\_type={ret\_type}\_fluct={fluct}\_rho={rho}\_classes={classes}\_type={Type}.jpg")  
  
plt.show()



fig, ax = plt.subplots(figsize=(15,8))  
  
ax.plot(close.index[window:n:tstep], for\_graph.values[window:n:tstep], label=f"{symbol\_for\_graph\_plot}", color='b')  
ax.set\_xlabel("time, days")  
ax.set\_ylabel(f"{symbol\_for\_graph\_plot}, RevDispEn")  
  
ax2 = ax.twinx()  
ax2.plot(close.index[window:n:tstep], RevDispEn, label="RevDispEn", color='r')  
  
fig.legend(loc="upper right", bbox\_to\_anchor=(1,1), bbox\_transform=ax.transAxes)  
  
plt.savefig(f"RevDispEn\_symbol={symbol}\_window={window}\_step={tstep}\_d\_e={m}\_tau={tau}\_series\_type={ret\_type}\_fluct={fluct}\_rho={rho}\_classes={classes}\_type={Type}.jpg")  
  
plt.show()



# 7. Лабораторна робота № 6

**Тема.** Фрактальний аналіз складних систем.

**Мета.** Навчитися використовувати методи монофрактального аналізу для дослідження нелінійних характеристик складних систем.

## 7.1 Теоретичні відомості

### 7.1.1 Означення фрактала

**Фракталами** називають геометричні об’єкти: лінії, поверхні, просторові тіла, що мають сильно шорстку поверхню або форму і характеризуються властивістю самоподібності. Слово фрактал походить від латинського слова *fractus* і перекладається як дробовий, ламаний. Самоподібність як основна характеристика фрактала означає, що він більш-менш однорідно змінюється при широкому діапазоні масштабів. Так, при збільшенні маленькі фрагменти фрактала виходять дуже схожими на великі. В ідеальному випадку така самопподібність призводить до того, що фрактальний об’єкт виявляється інваріантним щодо розтягувань, тобто йому, як кажуть, притаманна дилатаційна симетрія. Вона передбачає незмінність основних геометричних особливостей фрактала при зміні масштабу.

Очевидно, що фрактальні об’єкти реального світу не є нескінченно самоподібними й існує мінімальний масштаб , такий, що на масштабі властивість самоподібності зникає. Окрім цього, на достатньо великих масштабах довжин , де — характерний геометричний розмір об’єктів, ця властивість самоподібності також порушується. Тому властивості природніх фракталів розглядаються лише на масштабах , що задовільняють відношення . Такі обмеження природні, оскільки, коли ми приводимо в якості прикладу фракталу — ламану, негладку траєкторію броунівської частинки, то ми розуміємо, що цей образ представляє очевидну ідеалізацію. Справа в тому, що на малих масштабах приховується граничність маси і розмірів броунівської частнки, а також кінцевість часу зіткнення. При врахуванні цих обставин траєкторія броунівської частинки починає представляти гладку криву.

Варто зазначити, що властивість точної самоподібності характерна лише для **регулярних фракталів**. Якщо замість детермінованого способу побудови включити в алгоритм їхнього створення деякий елемент випадковості (як це буває, наприклад, у багатьох процесах диференційованого зростання кластерів, електричному пробої тощо), то виникають так звані **випадкові фрактали**. Основна їхня відмінність від регулярних полягає в тому, що властивості самоподібності є справедливими тільки після відповідного усереднення за всіма статистично незалежними реалізаціями об’єкта. При цьому збільшена частина фрактала не точно ідентична вихідному фрагменту, проте їхні статистичні характеристики збігаються.

### 7.1.2 Довжина берегової лінії

|  |
| --- |
| Рисунок 7.1: Визначення довжини берегової лінії між точками А та В |

Першочергово поняття фрактала у фізиці виникло у зв’язку із завданням про визначення довжини берегової лінії. Під час її вимірювання за наявною картою місцевості з’ясувалася цікава деталь — чим більш великомасштабна карта береться, тим довшою виявляється ця берегова лінія. Нехай, наприклад, відстань по прямій між розсташованими на береговій лінії точками А та В дорівнює R (див. [Рисунок 7.1](#fig-costline)). Тоді, щоб виміряти довжину берегової лінії між цима точками, ми розташуємо по берегу жорстко пов’язані один з одним вузол так, що відстань між сусідніми вузлами дорівнювала б, наприклад, км. Довжину берегової лінії в кілометрах між точками А і В ми приймемо тоді рівною числу вузлів мінус 1, помноженому на 10. Наступне вимірювання цієї довжини ми зробимо так само, але відстань між сусідніми вузлами зробимо вже рівною км.

Виявляється, що результат цих вимірювань буде різним. При зменшенні масштабу ми отримуватимемо все більші й більші значення довжини. На відміну від гладкої кривої, лінія морського узбережжя виявляється найчастіше настільки порізаною (аж до найменших масштабів), що зі зменшенням довжини ланки величина — довжина берегової лінії — не прагне до кінцевого межі, а збільшується за степеневим законом

де — деякий степеневий показник, котрий іменується **фрактальною розмірністю** берегової лінії. Чим більше значення , тим більш ломаною або деталізованою представляється ця берегова лінія. Походження залежності (1) має бути інтуїтивно зрозумілим: чим менший масштаб ми використовуємо, тим меньші деталі узбережжя будуть враховані і тим менший вклад вони внесуть у вимірювану довжину. Навпаки, збільшуючи масштаб, ми “розгортаємо” узбережжя, зменшуючи довжину .

Таким чином, ми бачимо, що для визначення довжини берегової лінії за допомогою жорсткого масштабу , необхідно зробити кроків, причому величина змінюється з так, що залежить від за законом . У результаті зі зменшенням масштабу довжина берегової лінії необмежено зростає. Ця обставина різко відрізняє фрактальну криву від звичайної гладкої кривої (типу кола, еліпса), для якої межа довжини апроксимованої ламаної , яка апроксимує, за наближення до нуля довжини її ланки є скінченною. У результаті для гладкої кривої її фрактальна розмірність , тобто збігається з топологічною.

### 7.1.3 Фрактальна розмірність множин

Вище було введено поняття про фрактальну розмірність берегової лінії. Дамо тепер загальне визначення цієї величини. Нехай — звичайна Евклідова розмірність простору, в якому розташований наш фрактальний об’єкт ( — лінія, — площина, — звичайний тривимірний простір). Покриємо тепер цей об’єкт цілком -мірними “кулями” радіуса 1. Припустимо, що нам потребувалося для цього не менше, ніж куль. Тоді, якщо за досить малих величина змінюється з за степеневим законом

тоді — називається **хаусдорфовою** або **фрактальною розмірністю** цього об’єкта. Очевидно, що ця формула еквівалентна відношеню , що використовувалось вище для визначення довжини берегової лінії.

Формулу (2) можна переписати у вигляді

Це відношення й слугує загальним визначенням фрактальної розмірності . У відповідності до нього величина представляє **локальну** характеристику досліджуваного об’єкта.

### 7.1.4 Процедури обчислення монофрактальних розмірностей

Наразі існує багато визначень та методів вимірювання фрактальної розмірності. Найпоширенішими одновимірними фрактальними розмірностями є розмірність Хаусдорфа, розмірність Хігучі, розмірність Петросяна та Коробчаста розмірність. Розмірність Хаусдорфа є найпростішою фрактальною розмірністю. Але її обчислювальна складність є високою, що ускладнює її практичне застосування. Коробкова розмірність є відносно простою, і фрактальну розмірність сигналу можна отримати, регулюючи розмір довжини сторони коробки. Тому вона є широко впізнаваємою та застосовуваною. Який показник фрактальної розмірності найточніше описує складність сигналу та здатний ідентифікувати кризові явища і представляє ключовий момент цієї лабораторної роботи.

#### 7.1.4.1 *R/S*-аналіз

Метод *R/S*-аналізу, розроблений Мандельбротом та Уоллесом, базується на попередньо створеному методі гідрологічного аналізу Херста, і дозволяє обчислювати параметр самоподібності , який вимірює інтенсивність довготривалих залежностей у часовому ряді. Коефіцієнт , який називають коефіцієнтом Херста, містить мінімальні прогнози стосовно природи системи, що вивчається, і може класифікувати часові ряди. За допомогою цього показника розрізняють випадкові (гаусові) та невипадкові ряди; окрім того, він пов’язаний із фрактальною розмірністю, що, у свою чергу, характеризує ступінь згладженості графіка, побудованого на основі часового ряду. Методом *R/S*-аналізу можливо також виявити максимальну довжину інтервалу (цикл), на якому значення зберігають інформацію про початкові дані системи (довготривала пам’ять).

Аналіз починається з побудови ряду логарифмічних прибутковостей, , де — значення вихідного часового ряду в момент , — часовий крок. Отримана послідовність розбивається на підпослідовностей довжини .

Для кожної підпослідовності :

1. шукається середнє значення та стандартне відхилення ;
2. дані нормалізуються шляхом віднімання середнього значення послідовності , ;
3. знаходиться кумулятивна сума послідовності ів: , ;
4. у межах кожної підпослідовності знаходиться розмах між максимальним та мінімальним значеннями: , який стандартизується середнім квадратичним відхиленням ;
5. обчислюється середнє нормованих значень розмаху для всіх підпослідовностей довжини .

*R/S*-статистика, обрахована таким чином, відповідає співвідношенню , де значення може бути отримане шляхом обчислення для послідовностей інтервалів зі збільшенням часового горизонту:

Знайти коефіцієнт Херста можна, побудувавши залежність vs.  у подвійному логарифмічному масштабі і взявши коефіцієнт нахилу прямої, яка інтерполює точки отриманого графіка. Якщо значення , говорять про послідовність, що представляє собою білий шум; свідчить про персистентний ряд, коли існує тенденція слідування великих значень ряду за великими і навпаки; вказує на антиперсистентний ряд.

При збільшенні часового горизонту коефіцієнт нахилу інтерполюючої прямої повинен прямувати до значення ; сам процес переходу свідчить про втрату впливу початкових умов на поточні значення, і, таким чином, можна говорити про горизонт довгої пам’яті — це точка, до якої коефіцієнт нахилу інтерполюючої прямої відмінний від 0.5, а після — близько 0.5.

|  |
| --- |
| Примітка до *R/S*-аналізу |
| Між фрактальною розмірністю та показником Херста також існує зв’язок  Якщо для берегової лінії ми визначали масштабування її довжини в залежності від зміни , то у випадку з *R/S*-аналізом ми визначаємо зміну нормованого розмаху значень ряду в межах масштабу . |

#### 7.1.4.2 Аналіз детрендованих флуктуацій

Аналіз детрендований флуктуацій (Detrended fluctuation analysis, DFA) базується на гіпотезі про те, що корельований часовий ряд може бути відображений на самоподібний процес шляхом інтегрування. Таким чином, вимірювання властивостей самоподібності може непрямо свідчити про кореляційні властивості ряду. Переваги АДФ порівняно з іншими методами (спектральний аналіз, *R/S*-аналіз) полягають в тому, що він виявляє довгочасові кореляції нестаціонарних часових рядів, а також дозволяє ігнорувати очевидні випадкові кореляції, що є наслідком нестаціонарності.

Існують DFA різних порядків, що відрізняються трендами, які вилучаються з даних.

Розглянемо DFA найнижчого порядку.

1. Для часового ряду довжини знаходиться кумулятивна сума, , де — це -те значення часового ряду, — його середнє значення, .
2. Отриманий ряд розбивається на підпослідовностей (вікон) однакової ширини і для кожної підпослідовності (у кожному вікні) виконується наступне:
   * за допомогою методу найменших квадратів знаходиться локальний лінійний тренд ;
   * підпослідовність детрендується шляхом віднімання значення локального тренду від значень ряду , що належать послідовності ;
   * знаходиться середнє детрендований значень.

Для отриманих таким чином значень на всіх підпослідовностях знаходиться:

де — кількість точок у підпослідовності (ширина вікна), — кількість підпослідовностей, — середнє детрендованих значень для підпослідовності .

Вказана процедура повторюється для різних значень , внаслідок чого ми отримує набір залежностей від . Побудова залежності від та інтерполяція отриманих значень лінією регресії дає змогу обчислити показник скейлінга , що є коефіцієнтом кута нахилу інтерполяційної прямої і характеризує зміну кореляцій флуктуацій часового ряду при збільшенні часового інтервалу .

Порівняно із *R/S*-аналізом, DFA дає більші можливості інтерпретації скейлінгового показника :

* для випадкового ряду (перемішаного чи “сурогатного”) ;
* при наявності лише короткочасових кореляцій може відрізнятись від 0.5, проте має тенденцію прямувати до 0.5 при збільшенні розміру вікна;
* Значення показує персистентні довгочасові кореляції, що відповідають степеневому закону;
* означає антиперсистентний ряд;
* cпеціальний випадок, коли , означає наявність шуму.
* для випадків, коли , кореляції існують, проте перестають відображувати степеневу залежність;
* випадок свідчить про Броунівський шум, інтегрований білий шум.

У випадку степеневої залежності функції автокореляцій спостерігається спад автокореляції з показником :

На додачу до цього, спектральна густина також спадає за степеневим законом:

Відповідні показники виражаються через наступні відношення:

* ;
* .

У DFA другого порядку (DFA2) обчислюються відхилення профілю від інтерполяційного многочлена другого порядку. Таким чином, вилучаються впливи можливих лінійних та параболічних трендів для масштабів, більших за розглядувані. Взагалі, у DFA порядку обчислюються відхилення профілю від інтерполяційного многочлена -го порядку, що вилучає вплив всіх можливих трендів порядків до () для масштабів, більших від розміру вікна.

Потім обчислюється найближчий поліном для профілю на кожному із сегментів і визначається відхилення

Далі знаходиться середнє значення флуктуацій всіх детрендованих профілів:

Значення формули (6) можна трактувати як середньоквадратичний зсув (переміщення) точки випадкових блукань у ланцюжку після кроків.

#### 7.1.4.3 Фрактальна розмірність Хігучі

Фрактальна розмірність Хігучі — це один з різновидів монофрактальної розмірності, яка визначається наступним чином:

Припустимо, що у нас є часовий ряд

і реконструйований часовий ряд :

для ; де представляє початковий час; представляють ступінь часового зміщення. Позначення представляє цілу частину . Для кожного реконструйованого часового ряду розраховується середня довжина часової послідовності :

Далі, для всіх середніх довжин , знаходиться загальне середнє:

Згідно методу Хігучі узагальне середнє значення пропорційне масштабу , тобто

Далі логарифмуємо обидві сторони й отримуємо наступну рівність:

Інтерполювавши лінію регресії через залежність від , ми можемо отримати показник фрактальності , отримавши кут нахилу цієї лінії. Показник і представлятиме фрактальну розмірність Хігучі.

#### 7.1.4.4 Фрактальна розмірність Петросяна

Спочатку, для часового ряду , створюємо його дискретизовану (бінарну) версію, :

Фрактальну розмірність Петросяна може бути визначена як

де — кількість загальних змін знаку змінної :

#### 7.1.4.5 Фрактальна розмірність Каца

Представимо, що сигнал складається з пари точок . Тоді, фрактальна розмірність Каца визначається як

де визначається наступним чином:

Значення визначається як максимальна відстань від початкової точки до всіх інших точок, а саме може бути розраховане наступним чином:

#### 7.1.4.6 Фрактальна розмірність Севчика

Спочатку, для множини значень виконується нормалізація:

Фрактальна розмірність Севчика може бути визначена як

де — це довжина сигналу, що може бути визначена як

#### 7.1.4.7 Фрактальна розмірність через нормалізовану щільність довжини

Даний показник розраховується в наступний спосіб:

1. Для часового ряду виконується стандартизація: , де — це середнє значення ряду, — це стандартне відхилення.
2. Розраховується нормалізована щільність довжини:

Фактичний розрахунок фрактальної розмірності сигналу базується на побудові монотонної калібрувальної кривої, , за набором функцій Вейєрштрасса, для яких значення задаються теоретично.

1. Для обчислювальних цілей необхідно створити математичну модель цієї залежності. Автори даного підходу тестували дві моделі:
   * логарифмічну модель:
   * степеневу модель: . Бібліотека neurokit2 використовує саме степеневу модель. Параметр , і , згідно статті *Kalauzi et al. 2009*.

#### 7.1.4.8 Фрактальна розмірність через нахил спектральної щільності потужності

Фрактальну розмірність можна обчислити на основі аналізу нахилу спектральної щільності потужності (power spectral density slope, PSD) в сигналах, що характеризуються частотною степеневою залежністю.

Спочатку виконується перетворення часового ряду до частотної області і далі сигнал розбивається на синусоїдальні та косинусоїдальні хвилі певної амплітуди, які разом “складаються”, щоб представити вихідний сигнал. Якщо існує систематичний зв’язок між частотами в сигналі і потужністю цих частот, то в логарифмічних координатах це проявляється в лінійній залежності. Кут нахилу лінії регресії приймається як оцінка фрактальної розмірності.

Нахил 0 відповідає білому шуму, а нахил менше 0, але більше -1, відповідає рожевому шуму, тобто шуму . Спектральні нахили крутіші за -2 вказують на дробовий броунівський рух, що є втіленням процесів випадкового блукання.

#### 7.1.4.9 Кореляційна розмірність

Кореляційна розмірність () — це похідна величина від кореляційного інтеграла Кореляційний інтеграл (кореляційна сума) може бути поданий в такому вигляді:

Сама кореляційна розмірність може бути виведена з наступної степеневої залежності:

або слідуючим чином:

де — це сумарна кількість пар точок, відстань між якими менша за радіус .

За формулою ми відбираємо -ту траєкторію та всі інші -ті траєкторії, і дивимося, чи потрапляють -ті траєкторії в округ -ої траєкторії з радіусом . Якщо відстань між ними не перевищує округ зі згаданим радіусом, ми ставимо 1. Але якщо відстань між траєкторіями більша за , тоді ставимо 0. Далі все це підсумовується, ділиться на загальну кількість траєкторій. По суті кореляційний інтеграл це середня ймовірність того, що дві розглянуті траєкторії фазового простору, що розглядаються, будуть знаходитися досить близько одна до одної. Чим тісніше розташовані точки фазового простору одна до одної, тим більше значення кореляційного інтеграла. Чим більш рівновіддаленими видаються траєкторії одна від одної, тим ближче значення кореляційного інтеграла до нуля.

Значення кореляційної розмірності ми можемо відшукати аналогічно попередним фрактальним показникам: ми шукаємо залежність кореляційного інтеграла від значення . Ця залежність будується в логарифмічному масштабі.

|  |
| --- |
| Додаткова інформація по кореляційні розмірності |
| Кореляційна розмірність за аналогією з попередніми показниками — це теж тангенс кута нахилу лінії регресії, побудованої в логарифмічному масштабі, але для залежності кореляційного інтеграла від . За аналогією з іншими показниками, кореляційна розмірність визначає швидкість зміни значення кореляційного інтеграла (крутість нахилу лінії регресії). |

|  |
| --- |
| Рисунок 7.2: Зміна значення кореляційного інтеграла в залежності від ступеня розкиданості точок по фазовому простору системи |

За фазовим простором цієї хмари точок ( [Рисунок 7.2](#fig-d2) ) можна бачити, що більша згуртованість точок одна до одної має асоціюватися з меншою кореляційною розмірністю. Рисунок (а) характеризується найвищою близькістю точок одна до одної, але при цьому найменшою кореляційною розмірністю. Це можна пояснити так: якщо ми будуємо сітку з кіл радіусом і поступово її збільшуємо, ми вже перестаємо бачити нові прилеглі траєкторії до тих -их траєкторій, які ми розглядали із самого спочатку, на початкових значеннях . На рисунках (b) і (c) видно, що хмара траєкторій фазового простору є більш рівномірно розподіленою. За поступового збільшення радіуса кіл із центрами в кожній -ій траєкторії ми повинні спостерігати пропорційне збільшення значення кореляційного інтеграла. У даному випадку при поступовому збільшенні ми спостерігаємо появу все більшої і більшої кількості точок.

У періодичних системах кореляційна розмірність залишається постійною і дорівнює розмірності вкладення. Наприклад, для простої періодичної системи, такої як синусоїда, кореляційна розмірність дорівнюватиме 1 (оскільки вона лежить на одновимірній кривій), а якщо систему реконструюють у двовимірному фазовому просторі (за двома координатами), то кореляційна розмірність дорівнюватиме 2. У таких системах кореляційна розмірність не змінюється.

Для хаотичних систем кореляційна розмірність має характерну поведінку, яка залежить від кількості змінних (розмірностей), необхідних для точного опису динаміки системи. На відміну від періодичних систем, кореляційна розмірність зростає в міру збільшення розмірності фазового простору, поки не досягне плато.

**Електрокардіограма** (ЕКГ): ЕКГ-сигнали відображають електричну активність серця. Складність ЕКГ-сигналу може бути оцінена за допомогою кореляційної розмірності. Очікується, що кореляційна розмірність ЕКГ здорового серця буде вищою через наявність складних патернів і варіабельності. З іншого боку, аномальні ЕКГ-сигнали, наприклад, від пацієнтів з аритміями або серцевими захворюваннями, можуть мати нижчу кореляційну розмірність через втрату складності сигналу.

**Електроенцефалограма** (ЕЕГ): Сигнали ЕЕГ реєструють електричну активність мозку. Кореляційна розмірність може використовуватися для аналізу складності мозкової активності, яка може змінюватися залежно від різних когнітивних станів, стадій сну або неврологічних розладів. У здорових людей сигнали ЕЕГ у стані бадьорості та уваги можуть мати вищу кореляційну розмірність порівняно з сигналами у стадії сну, коли активність мозку є більш регулярною і синхронізованою.

**Дихальні сигнали**: Дихальні сигнали, такі як частота дихання або повітряний потік, також можуть бути проаналізовані за допомогою кореляційної розмірності. Складність цих сигналів може змінюватися залежно від таких факторів, як стрес, фізичне навантаження або наявність респіраторних захворювань. За нормального дихання може спостерігатися вища кореляційна розмірність, тоді як порушення в дихальних сигналах, наприклад, за обструктивного апное уві сні або дихальних розладів, можуть призвести до зниження кореляційної розмірності.

**Аналіз ходи**: Кореляційна розмірність може бути використана для аналізу моделей ходи. Вона може допомогти в розумінні складності рухів людини під час ходьби або бігу. Зміни в кореляційній розмірності сигналів ходи можуть свідчити про зміну стабільності ходи або про наявність відхилень у ході, викликаних неврологічними або опорно-руховими захворюваннями.

**Варіативність динаміки серцевого ритму** (ВСР): ВСР являє собою зміну часових інтервалів між послідовними ударами серця. Вона перебуває під впливом вегетативної нервової системи і відображає адаптивність і складність серцево-судинної системи. Вищий рівень ВСР, що відповідає вищій кореляційній розмірності, зазвичай асоціюється з кращим станом серцево-судинної системи та її адаптивністю до фізіологічних змін і змін навколишнього середовища. Її падіння може асоціюватися з аномальною динамікою серця.

**Послідовності ДНК**: Кореляційна розмірність може бути використана і при аналізі послідовностей ДНК. Вона допомагає виявити самоподібні або фрактальні патерни всередині послідовностей, що може мати значення для розуміння генетичної складності, еволюційних зв’язків і регуляції генів. Висока кореляційна розмірність — висока складність ланцюжка ДНК. Мала кореляційна розмірність — спрощений ланцюжок ДНК.

**Фінансові ринки** Вища кореляційна розмірність у даних часових рядів фінансового ринку свідчить про більшу складність та існування в їхній основі самоподібних моделей або фрактальних структур. Хаотична поведінка цін на акції може бути пов’язана з періодами високої волатильності та непередбачуваності. З іншого боку, нижча величина кореляційної розмірності може свідчити про більш передбачувані та менш складні рухи цін, що відповідає періодам стабільності або менш волатильним ринковим умовам.

|  |
| --- |
| Примітка по кореляційні розмірності |
| Кореляційна розмірність розглядає кількість інформації (кубиків, кіл), необхідну для опису **тільки пари точок** у фазовому просторі. |

## 7.2 Хід роботи

Розглянемо як можна застосовувати зазначені показники в якості індикаторів кризових станів.

Спочатку імпортуємо необхідні бібліотеки для подальшої роботи:

import matplotlib.pyplot as plt   
import numpy as np  
import neurokit2 as nk  
import yfinance as yf  
import pandas as pd  
import scienceplots  
from tqdm import tqdm  
  
%matplotlib inline

Далі виконаємо налаштування формату виведення рисунків:

plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься  
 # для виведення рисунків  
  
params = {  
 'figure.figsize': (8, 6), # встановлюємо ширину та висоту рисунків за замовчуванням  
 'font.size': 22, # розмір фонтів рисунку  
 'lines.linewidth': 2, # товщина ліній  
 'axes.titlesize': 'small', # розмір титулки над рисунком  
 "font.family": "sans-serif", # сімейство стилів підписів   
 "font.serif": ["Times"], # стиль підпису  
 'savefig.dpi': 300 # якість збережених зображень  
}  
  
plt.rcParams.update(params) # оновлення стилю згідно налаштувань

У даній роботі скористаємось монофрактальними показниками для ідентифікації кризових явищ на ринку золота. Розглянемо значення золота за весь період, що представляє Yahoo! Finance. Для цього нам не треба буде вказувати початкову та кінцеву дати:

symbol = 'GC=F' # Символ індексу  
  
data = yf.download(symbol) # вивантажуємо дані  
time\_ser = data['Adj Close'].copy() # зберігаємо саме ціни закриття  
  
xlabel = 'time, days' # підпис по вісі Ох   
ylabel = symbol # підпис по вісі Оу

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

Виводимо досліджуваний ряд:

fig, ax = plt.subplots() # Створюємо порожній графік  
ax.plot(time\_ser.index, time\_ser.values) # Додаємо дані до графіку  
ax.legend([symbol]) # Додаємо легенду  
ax.set\_xlabel(xlabel) # Встановимо підпис по вісі Ох  
ax.set\_ylabel(ylabel) # Встановимо підпис по вісі Oy  
  
plt.xticks(rotation=45) # оберт позначок по осі Ох на 45 градусів  
  
plt.savefig(f'{symbol}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 7.3: Динаміка щоденних змін індексу золота |

Визначимо функцію transformation() для стандартизації ряду:

def transformation(signal, ret\_type):  
  
 for\_rec = signal.copy()  
  
 if ret\_type == 1: # Зважаючи на вид ряду, виконуємо  
 # необхідні перетворення  
 pass  
 elif ret\_type == 2:  
 for\_rec = for\_rec.diff()  
 elif ret\_type == 3:  
 for\_rec = for\_rec.pct\_change()  
 elif ret\_type == 4:  
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 elif ret\_type == 5:   
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 for\_rec = for\_rec.abs()  
 elif ret\_type == 6:  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
  
 for\_rec = for\_rec.dropna().values  
  
 return for\_rec

### 7.2.1 Обчислення показника Херста із використанням *R/S*-аналізу

Для подальших розрахунків використовуватимемо бібліотеку neurokit2 та fathon. Другу можна встановити в наступний спосіб:

!pip install fathon

Далі імпортуємо саму бібліотеку та дотичні до неї модулі:

import fathon  
from fathon import fathonUtils as fu

Бібліотека neurokit містить необхідний метод для *R/S*-аналізу — fractal\_hurst. Його синтаксис виглядає наступним чином:

**fractal\_hurst(signal, scale='default', corrected=True, show=False)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень або датафрейму бібліотеки pandas.
* **scale** (*list*) — список, що містить довжини вікон (кількість точок даних у кожній підмножині ряду), на які розбито сигнал.
* **corrected** (*bool*) — якщо значення True, до вихідних даних буде застосовано поправочний коефіцієнт Аніса-Ллойда-Пітерса відповідно до очікуваного значення для окремих значень (*R/S*).
* **show** (*bool*) — якщо значення True, виводить залежність від (**scale**) у подвійному логарифмічному масштабі.

**Повертає**

* **h** (*float*) — показник Херста.
* \*\*kwargs — словник, що містить інформацію відносно використовуваних у процедурі параметрів.

Розглянемо ступінь трендостійкості в динаміці фондового індексу золота, використовуючи весь часовий ряд. Далі знайдемо значення показника Херста в рамках віконної процедури.

#### 7.2.1.1 Увесь часовий ряд

Першочергово знайдемо значення прибутковостей для нашого ряду та стандартизуємо їх. Після цього виконаємо обчислення.

signal = time\_ser.copy()  
ret\_type = 4 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
for\_rs = transformation(signal, ret\_type)

Виконуємо *R/S*-аналіз:

h, info = nk.fractal\_hurst(for\_rs, corrected=False, show=True)

|  |
| --- |
| Рисунок 7.4: Залежність значень R/S від скейлінгу побудованих в логарифмічному масштабі |

Як ми можемо бачити з [Рисунок 7.4](#fig-gold-rs-all), значення , що свідчить про подібність динаміки золота до випадкового блукання. Але оскільки закони, що регулюють ринок, змінюються з часом, мають змінюватись і кореляції всередині системи, а одже коефіцієнт Херста також має залежати від періоду в якому він розглядається.

#### 7.2.1.2 Віконна процедура

Визначимо функцію для побудови парних графіків:

def plot\_pair(x\_values,   
 y1\_values,  
 y2\_values,   
 y1\_label,   
 y2\_label,  
 x\_label,   
 file\_name, clr="magenta"):  
  
 fig, ax = plt.subplots()  
  
 ax2 = ax.twinx()  
  
 ax2.spines.right.set\_position(("axes", 1.03))  
  
 p1, = ax.plot(x\_values,   
 y1\_values,   
 "b-", label=fr"{y1\_label}")  
 p2, = ax2.plot(x\_values,  
 y2\_values,   
 color=clr,   
 label=y2\_label)  
  
 ax.set\_xlabel(x\_label)  
 ax.set\_ylabel(f"{y1\_label}")  
  
 ax.yaxis.label.set\_color(p1.get\_color())  
 ax2.yaxis.label.set\_color(p2.get\_color())  
  
 tkw = dict(size=2, width=1.5)  
  
 ax.tick\_params(axis='x', \*\*tkw)  
 ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
 ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
  
  
 ax2.legend(handles=[p1, p2])  
  
 plt.savefig(file\_name + ".jpg")  
   
 plt.show();

Приступимо до віконної процедури:

# встановлюємо параметри  
ret\_type = 4 # вид ряду  
window = 250 # ширина вікна  
tstep = 1 # часовий крок вікна   
length = len(time\_ser.values) # довжина самого ряду  
corr = False # поправочний коефіцієнт Аніса-Ллойда-Пітерса  
  
H = [] # масив для віконного Херсту

for i in tqdm(range(0,length-window,tstep)): # фрагменти довжиною window   
 # з кроком tstep  
  
 # відбираємо фрагменти  
 fragm = time\_ser.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо взаємну інформацію   
 h, \_ = nk.fractal\_hurst(fragm, corrected=corr, show=False)  
   
 # та додаємо результат до масиву значень  
 H.append(h)

1%| | 41/5531 [00:00<00:13, 404.03it/s]

100%|██████████| 5531/5531 [00:13<00:00, 419.90it/s]

np.savetxt(f"rs\_hurst\_name={symbol}\_window={window}\_step={tstep}\_ \  
 rettype={ret\_type}\_corrected={corr}.txt" , H)

Візуалізуємо результат:

measure\_label = r'$H$'  
file\_name = f"rs\_hurst\_name={symbol}\_window={window}\_step={tstep}\_ \  
 rettype={ret\_type}\_corrected={corr}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 H,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name)

|  |
| --- |
| Рисунок 7.5: Динаміка індексу золота та показника Херста |

На представленому рисунку ( [Рисунок 7.5](#fig-gold-wind-rs) ) можемо бачити, що показник Херста зростає в передкризовий період та спадає під час кризи. Перед кризою динаміка ринку характеризується зростанням трендостійкості (персистентності), що відзеркалює зростання скорельованості дій між трейдерами ринку.

### 7.2.2 Обчислення на основі DFA

Бібліотека fathon представляє інструментарій як для виконання класичного аналізу детрендованих флуктуацій, так і для його мультифрактального аналогу, мова про який піде в наступній лабораторній.

#### 7.2.2.1 Для всього ряду

Спочатку представимо значення для всього ряду. Процедура розрахунків на основі бібліотеки fathon виглядатиме наступним чином:

1. знаходимо стандартизовані прибутковості ряду

signal = time\_ser.copy()  
ret\_type = 4 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
for\_dfa = transformation(signal, ret\_type)

cumulat = fu.toAggregated(for\_dfa) # знаходимо кумулятивні накопичення  
  
rev = True # чи повторювати розрахунок ф-ції флуктуацій з кінця  
order = 2 # порядок локального лінійного тренду   
  
pydfa = fathon.DFA(cumulat) # ініціалізація об'єкту DFA  
 # для виконання подальших обчислень  
  
win\_beg = 100 # початкова ширина сегментів  
win\_end = 2000 # кінцева ширина сегментів  
  
wins = fu.linRangeByStep(win\_beg, win\_end) # генеруємо масив   
 # лінійно розділених   
 # елементів.  
  
n, F = pydfa.computeFlucVec(wins,   
 polOrd=order,   
 revSeg=rev) # знаходимо функцію флуктуацій  
  
H, H\_intercept = pydfa.fitFlucVec() # знаходимо показник альфа

Виводимо залежність функції флуктуацій від характеристичного масштабу:

polyfit = np.polyfit(np.log(n), np.log(F), 1)  
fluctfit = np.exp(1) \*\* np.polyval(polyfit, np.log(n))

Будуємо залежність функції флуктуацій від масштабу в подвійному логарифмічному масштабі:

fig, ax = plt.subplots()  
fig.suptitle("Показник Херста на основі DFA")  
  
ax.scatter(  
 np.log(n),  
 np.log(F),  
 marker="o",  
 zorder=1,  
 label="\_no\_legend\_",  
 )  
  
label = fr"$\alpha$ = {H:.2f}"  
ax.plot(np.log(n), np.log(fluctfit),   
 color="#E91E63", zorder=2,   
 linewidth=3, label=label)  
  
ax.set\_ylabel(r'$\ln{F\_{2}(n)}$')  
ax.set\_xlabel(r'$\ln{n}$')  
  
ax.legend(loc="lower right")  
  
plt.show()

|  |
| --- |
| Рисунок 7.6: Логарифмічна залежність значень функції флуктуацій від скейлінгу |

Процедура DFA показує, що значення індексу золота представляються скоріше антиперсистентними, але представлений результат доволі близький до того, що був отриманий за допомогою *R/S*-аналізу. Розглянемо значення в рамках алгоритму ковзного вікна.

#### 7.2.2.2 Віконна процедура

Визначимо наступні параметри:

window = 250 # розмір вікна  
tstep = 1 # крок вікна  
ret\_type = 4 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням   
 # та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
rev = True # чи повторювати розрахунок ф-ції флуктуацій з кінця  
order = 2 # порядок поліноміального тренду  
  
periods = 1  
  
win\_beg = 10 # початковий масштаб сегментів  
win\_end = window-1 # кінцевий масштаб сегментів  
  
  
  
length = len(time\_ser.values) # довжина ряду  
  
alpha = [] # масив показників альфа (Херста)  
D\_f = [] # фрактальна розмірність  
beta = [] # показник спектральної щільності  
gamma = [] # показник автокореляції

Знайдемо показник Херста (), фрактальну розмірність (), показник спектральної щільності () та показник автокореляції ():

for i in tqdm(range(0,length-window,tstep)):  
   
 # відбираємо фрагменти  
 fragm = time\_ser.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # знаходимо кумулятивні накопичення  
 cumulat\_wind = fu.toAggregated(fragm)   
  
 # ініціалізація об'єкту DFA  
 pydfa = fathon.DFA(cumulat\_wind)   
  
 # генеруємо масив лінійно розділених елементів  
 wins = fu.linRangeByStep(win\_beg, win\_end)   
  
 # знаходимо функцію флуктуацій  
 n, F\_wind = pydfa.computeFlucVec(wins, polOrd=order, revSeg=rev)   
  
 # знаходимо показник альфа  
 H\_wind, \_ = pydfa.fitFlucVec()  
  
 # знаходимо фрактальну розмірність   
 D = 2. - H\_wind  
  
 # показник спектральної щільності  
 bi = 2. \* H\_wind - 1   
  
 # показник автокореляції  
 gi = 2. - 2. \* H\_wind  
  
 alpha.append(H\_wind)  
 D\_f.append(D)  
 beta.append(bi)  
 gamma.append(gi)

100%|██████████| 5531/5531 [00:57<00:00, 96.13it/s]

Зберігаємо абсолютні значення показників до текстових файлів:

np.savetxt(f"alpha\_{symbol}\_{window}\_{tstep}\_ \  
 {ret\_type}\_{order}\_{win\_beg}\_{win\_end}.txt", alpha)  
np.savetxt(f"D\_f\_{symbol}\_{window}\_{tstep}\_ \  
 {ret\_type}\_{order}\_{win\_beg}\_{win\_end}.txt", D\_f)  
np.savetxt(f"beta\_{symbol}\_{window}\_{tstep}\_ \  
 {ret\_type}\_{order}\_{win\_beg}\_{win\_end}.txt", beta)  
np.savetxt(f"gamma\_{symbol}\_{window}\_{tstep}\_ \  
 {ret\_type}\_{order}\_{win\_beg}\_{win\_end}.txt", gamma)

Оголошуємо мітки для рисунків та назви збережених рисунків:

label\_alpha = fr'$\alpha$'  
label\_d = fr'$D\_f$'  
label\_beta = fr'$\beta$'  
label\_gamma = fr'$\gamma$'  
  
file\_name\_alpha = f"alpha\_{symbol}\_{window}\_{tstep}\_ \  
 {ret\_type}\_{order}\_{win\_beg}\_{win\_end}"  
file\_name\_d = f"D\_f\_{symbol}\_{window}\_{tstep}\_ \  
 {ret\_type}\_{order}\_{win\_beg}\_{win\_end}"  
file\_name\_beta = f"beta\_{symbol}\_{window}\_{tstep}\_ \  
 {ret\_type}\_{order}\_{win\_beg}\_{win\_end}"  
file\_name\_gamma = f"gamma\_{symbol}\_{window}\_{tstep}\_ \  
 {ret\_type}\_{order}\_{win\_beg}\_{win\_end}"

Виводимо результати:

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 alpha,   
 ylabel,   
 label\_alpha,  
 xlabel,  
 file\_name\_alpha)

|  |
| --- |
| Рисунок 7.7: Динаміка індексу золота та показника альфа |

Якщо порівнювати з *R/S*-аналізом, [Рисунок 7.7](#fig-gold-wind-alpha) демонструє, що динаміка *узагальненого показника Херста* отриманого за допомогою DFA є набагато стабільнішою. Тепер ми здатні диференціювати значну частку крахових подій, що мали місце на ринку золота. Узагальнений Херст показує, що передкризові явища характеризуються зростанням трендостійкості ринку або, іншими словами, підвищенням ступеня самоорганізації системи.

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 D\_f,   
 ylabel,   
 label\_d,  
 xlabel,  
 file\_name\_d)

|  |
| --- |
| Рисунок 7.8: Динаміка індексу золота та фрактальної розмірності |

[Рисунок 7.8](#fig-gold-wind-df) показує, що характеризується спадом при кризових станах. Це є індикатором того, що вищий ступень організованості ринку відзеркалюється в більш згладженій або менш шорстких флуктуаціях досліджуваного сигналу.

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 beta,   
 ylabel,   
 label\_beta,  
 xlabel,  
 file\_name\_beta)

|  |
| --- |
| Рисунок 7.9: Динаміка індексу золота та показника спектральної щільності |

На даному рисунку ми бачимо динаміку показника , що відноситься до спектральної густини потужності (), зростає в кризові періоди, що говорить про спад потужності сигналу, що припадає на одиничний інтервал частоти. Це також є свідченням зростання кореляційних властивостей системи.

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 gamma,   
 ylabel,   
 label\_gamma,  
 xlabel,  
 file\_name\_gamma)

|  |
| --- |
| Рисунок 7.10: Динаміка індексу золота та показника автокореляції |

На [Рисунок 7.10](#fig-gold-wind-gamma) видно, що показник спадає в кризові та передкризові періоди. Це є показником сповільнення спаду функції автокореляції, що в свою чергу також вказує на зростання корельованності динаміки системи.

### 7.2.3 Обчислення фрактальної розмірності Хігучі

Як уже зазначалося, фрактальна розмірність Хігучі є одним із різновидів фрактальної розмірності для часових рядів. Вона обчислюється шляхом реконструкції -максимальної кількості нових наборів даних. Для кожного відновленого набору даних обчислюється довжина кривої і відкладається проти відповідного k-значення в логарифмічній шкалі. HFD відповідає нахилу лінійного тренду за методом найменших квадратів.

Розрахуємо оптимальне значення для всього часового ряду. Бібліотека neurokit2 представляє готову процедуру для автоматизованого підбору даного параметру. Оптимальний розраховується на основі точки, в якій значення фрактальної розмірності досягає плато для діапазону значень (див. Vega, 2015).

Синтаксис даної функції виглядає наступним чином:

**complexity\_k(signal, k\_max='max', show=False)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* (*Union[int, str, list], optional*) — максимальна кількість інтервалів (має бути більше або дорівнювати 3), які потрібно перевірити. Якщо =default, тоді вибирається максимально можливе значення, що відповідає половині довжини сигналу.
* **show** (*bool*) — візуалізовується нахил кривої для обраного значення .

**Повертає**

* **k** (*float*) — оптимальний часового ряду.
* **info** (*dict*) — словник, що містить додаткову інформацію про параметри, які використовуються для обчислення оптимального .

#### 7.2.3.1 Для всього ряду

Для подальших розрахунків спочатку виконаємо перетворення ряду. Будемо використовувати вихідний часовий ряд для подальших розрахунків:

signal = time\_ser.copy()  
ret\_type = 1 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
for\_higuchi = transformation(signal, ret\_type)

І тепер отримаємо оптимальне значення згідно зазначеної процедури:

k\_max, info = nk.complexity\_k(for\_higuchi, k\_max=100, show=True)

|  |
| --- |
| Рисунок 7.11: Залежність розмірності Хігучі від діапазону значень kmax |

Тепер побудуємо залежність довжини сигналу від часового зміщення в логарифмічному масштабі. Для фрактального сигналу має зберігатися лінійна залежність. Бібліотека neurokit2 містить метод для розрахунку даної фрактальної розмірності. Синтаксис цієї процедури виглядає наступним чином:

**fractal\_higuchi(signal, k\_max='default', show=False, \*\*kwargs)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* (*Union[int, str, list], optional*) — максимальна кількість інтервалів (має бути більше або дорівнювати 3), які потрібно перевірити.
* **show** (*bool*) — візуалізовується нахил кривої для обраного значення .

**Повертає**

* **HFD** (*float*) — фрактальна розмірність Хігучі для досліджуваного часового ряду.
* **info** (*dict*) — словник, що містить додаткову інформацію про параметри, які використовуються для обчислення фрактальної розмірності Хігучі.

hfd, info = nk.fractal\_higuchi(for\_higuchi, k\_max=k\_max, show=True)

|  |
| --- |
| Рисунок 7.12: Залежність довжини сигналу від часового зміщення |

У подальшому будемо послуговуватись отриманим оптимальним значенням для розрахунку розмірності Хігучі в рамках алгоритму ковзного вікна.

#### 7.2.3.2 Віконна процедура

Скористаємось наступними параметрами:

window = 250 # розмір вікна  
tstep = 1 # крок вікна  
ret\_type = 1 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням   
 # та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
k\_max\_wind = 30 # максимальне часове зміщення  
  
length = len(time\_ser.values) # довжина ряду  
  
hfd\_wind = [] # масив показників Хігучі

Розпочинаємо віконну процедуру:

for i in tqdm(range(0,length-window,tstep)):  
   
 # відбираємо фрагменти  
 fragm = time\_ser.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо фрактальну розмірність Хігучі  
 higuchi, \_ = nk.fractal\_higuchi(fragm,   
 k\_max=k\_max\_wind,   
 show=False)  
  
 # зберігаємо результат до масиву значень  
 hfd\_wind.append(higuchi)

100%|██████████| 5531/5531 [00:15<00:00, 362.90it/s]

Зберігаємо вихідні значення до текстового документа:

np.savetxt(f"fd\_higuchi\_name={symbol}\_kmax={k\_max\_wind}\_\  
 wind={window}\_step={tstep}.txt", hfd\_wind)

Оголошуємо мітки для рисунків та назви збережених рисунків:

label\_higuchi = fr'$HFD$'  
  
file\_name\_higuchi = f"fd\_higuchi\_name={symbol}\_kmax={k\_max\_wind}\_\  
 wind={window}\_step={tstep}"

Виводимо результат:

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 hfd\_wind,   
 ylabel,   
 label\_higuchi,  
 xlabel,  
 file\_name\_higuchi)

|  |
| --- |
| Рисунок 7.13: Динаміка індексу золота та фрактальної розмірності Хігучі |

Як можна бачити з представленого рисунку, фрактальна розмірність Хігучі може працювати як індикатор або індикатор-передвісник кризових явищ. Видно, що даний показник починає спадати у передкризові періоди чи у сам момент кризи, вказуючи на зростання згладженності динаміки системи, ступеня кореляцій та трендостійкості динаміки ринку.

### 7.2.4 Обчислення фрактальної розмірності Петросяна

Петросян (1995) запропонував швидкий метод оцінки фрактальної розмірності шляхом перетворення сигналу в двійкову послідовність, з якої оцінюється фрактальна розмірність. Існує кілька варіацій алгоритму (neurokit2, наприклад, пропонує "A", "B", "C" або "D"), що відрізняються насамперед способом створення дискретної (символьної) послідовності (див. complexity\_symbolize() для деталей). Найпоширеніший метод ("C", за замовчуванням) бінаризує сигнал за знаком послідовних різниць.

Більшість з цих методів дискретизації припускають, що сигнал є періодичним (без лінійного тренду). Для усунення лінійних трендів може бути корисним лінійне детрендування.

Синтаксис даної процедури має наступний вигляд:

**fractal\_petrosian(signal, symbolize='C', show=False)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **symbolize** (*str*) - метод перетворення неперервного вхідного сигналу в символьний (дискретний) сигнал. За замовчуванням присвоює 0 та 1 значенням нижче та вище середнього. Може мати значення None, що припускає, що вхідний сигнал вже є дискретним.
* **show** (*bool*) — виводить дискретизацію сигналу.

**Повертає**

* **PFD** (*float*) — фрактальна розмірність Петросяна для досліджуваного часового ряду.
* **info** (*dict*) — словник, що містить додаткову інформацію про параметри, які використовуються для обчислення фрактальної розмірності Петросяна.

Ми не розглядатимемо детально синтаксис функції complexity\_symbolize(). Опишемо лише ті методи дискретизації, що дотичні до фрактальної розмірності Петросяна: - **Метод A** бінаризує сигнал за більшими та меншими значеннями порівняно із середнім значенням сигналу. Еквівалентом є method="mean" (method="median" також є допустимим). - **Метод B** використовує значення, що знаходяться в діапазоні , проти значень, що виходять за межі цього діапазону. - **Метод C** обчислює різницю між послідовними вибірками та бінаризує їх залежно від знаку. - **Метод D** відокремлює послідовні відліки, що перевищують сигналу, від інших менших змін.

Тепер розглянемо віконну динаміку даного показника.

#### 7.2.4.1 Віконна процедура

Оскільки більшість з даних методів дискретизації вимагаються детрендування ряду, тоді будемо виконувати розрахунки для прибутковостей індексу золота.Скористаємось наступними параметрами:

window = 250 # розмір вікна  
tstep = 1 # крок вікна  
ret\_type = 4 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням   
 # та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
symb = "B" # тип дискретизації ряду  
  
length = len(time\_ser.values) # довжина ряду  
  
petr\_wind = [] # масив показників Петросяна

Розпочинаємо віконну процедуру:

for i in tqdm(range(0,length-window,tstep)):  
   
 # відбираємо фрагменти  
 fragm = time\_ser.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо фрактальну розмірність Петросяна  
 petrocian, \_ = nk.fractal\_petrosian(fragm,   
 symbolize=symb,   
 show=False)  
  
 # зберігаємо результат до масиву значень  
 petr\_wind.append(petrocian)

100%|██████████| 5531/5531 [00:05<00:00, 1034.18it/s]

Зберігаємо вихідні значення до текстового документа:

np.savetxt(f"fd\_petrosian\_name={symbol}\_method={symb}\_\  
 wind={window}\_step={tstep}.txt", petr\_wind)

Оголошуємо мітки для рисунків та назви збережених рисунків:

label\_petrocian = fr'$PFD$'  
  
file\_name\_petrocian = f"fd\_petrosian\_name={symbol}\_method={symb}\_\  
 wind={window}\_step={tstep}"

Виводимо результат:

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 petr\_wind,   
 ylabel,   
 label\_petrocian,  
 xlabel,  
 file\_name\_petrocian)

|  |
| --- |
| Рисунок 7.14: Динаміка індексу золота та фрактальної розмірності Петросяна |

[Рисунок 7.14](#fig-gold-wind-petro) показує, що показник Петросяна також спадає під час кризових подій, що вказує на зростання періодизації ринку та синхронізації активності трейдерів у відповідні моменти часу.

### 7.2.5 Обчислення фрактальної розмірності Каца

Тепер обчислюємо фрактальну розмірність Каца. Евклідові відстані між послідовними точками сигналу підсумовуються і усереднюються, а також визначається максимальна відстань між початковою точкою і будь-якою іншою точкою у вибірці.

Фрактальна розмірність варіюється від 1.0 для прямих ліній, приблизно до 1.15 для випадкових блукань і наближається до 1.5 для найбільш “дивних” форм сигналу.

Синтаксис процедури для розрахунку даної розмірності виглядає наступним чином:

**fractal\_katz(signal)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.

**Повертає**

* **KFD** (*float*) — фрактальна розмірність Каца для досліджуваного часового ряду.
* **info** (*dict*) — словник, що містить додаткову інформацію (наразі порожній, але повертається для узгодженості з іншими функціями).

#### 7.2.5.1 Віконна процедура

Оскільки даний показник є параметронезалежним, нам достатньо буде лише розміру часового вікна, кроку та типу ряду:

window = 250 # розмір вікна  
tstep = 1 # крок вікна  
ret\_type = 1 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням   
 # та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
length = len(time\_ser.values) # довжина ряду  
  
kz\_wind = [] # масив показників Каца

Розпочинаємо віконну процедуру:

for i in tqdm(range(0,length-window,tstep)):  
   
 # відбираємо фрагменти  
 fragm = time\_ser.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо фрактальну розмірність Хігучі  
 katz, \_ = nk.fractal\_katz(fragm)  
  
 # зберігаємо результат до масиву значень  
 kz\_wind.append(katz)

100%|██████████| 5531/5531 [00:01<00:00, 3412.20it/s]

Зберігаємо вихідні значення до текстового документа:

np.savetxt(f"fd\_katz\_name={symbol}\_wind={window}\_step={tstep}.txt", kz\_wind)

Оголошуємо мітки для рисунків та назви збережених рисунків:

label\_katz = fr'$KFD$'  
  
file\_name\_katz = f"fd\_katz\_name={symbol}\_wind={window}\_step={tstep}"

Виводимо результат:

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 kz\_wind,   
 ylabel,   
 label\_katz,  
 xlabel,  
 file\_name\_katz)

|  |
| --- |
| Рисунок 7.15: Динаміка індексу золота та фрактальної розмірності Каца |

На представленому рисунку видно, що фрактальна розмірність Каца також спадає у кризові та передкризові періоди. Це також є індикаторов зростання ступеня корельованості системи в дані періоди.

### 7.2.6 Обчислення фрактальної розмірності Севчика

Алгоритм цієї фрактальної розмірності був запропонований для обчислення фрактальної розмірності сигналів Севчиком (1998). Цей метод можна використовувати для швидкого вимірювання складності та випадковості сигналу.

Синтаксис методу, що ми використовуватимемо в подальшому, має наступний вигляд:

**fractal\_sevcik(signal)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.

**Повертає**

* **SFD** (*float*) — фрактальна розмірність Севчика для досліджуваного часового ряду.
* **info** (*dict*) — словник, що містить додаткову інформацію (наразі порожній, але повертається для узгодженості з іншими функціями).

#### 7.2.6.1 Віконна процедура

Для цього показника нам також не потребується нічого зайвого:

window = 250 # розмір вікна  
tstep = 1 # крок вікна  
ret\_type = 1 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням   
 # та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
length = len(time\_ser.values) # довжина ряду  
  
sfd\_wind = [] # масив показників Севчика

Розпочинаємо віконну процедуру:

for i in tqdm(range(0,length-window,tstep)):  
   
 # відбираємо фрагменти  
 fragm = time\_ser.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо фрактальну розмірність Севчика  
 sevcik, \_ = nk.fractal\_sevcik(fragm)  
  
 # зберігаємо результат до масиву значень  
 sfd\_wind.append(sevcik)

100%|██████████| 5531/5531 [00:01<00:00, 3440.38it/s]

Зберігаємо вихідні значення до текстового документа:

np.savetxt(f"fd\_cevcik\_name={symbol}\_wind={window}\_step={tstep}.txt", sfd\_wind)

Оголошуємо мітки для рисунків та назви збережених рисунків:

label\_sevcik = fr'$SFD$'  
  
file\_name\_sevcik = f"fd\_cevcik\_name={symbol}\_wind={window}\_step={tstep}"

Виводимо результат:

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 sfd\_wind,   
 ylabel,   
 label\_sevcik,  
 xlabel,  
 file\_name\_sevcik)

|  |
| --- |
| Рисунок 7.16: Динаміка індексу золота та фрактальної розмірності Севчика |

Бачимо, що і фрактальна розмірність Севчика реагує спадом на крахові події на ринку золота. Особливо характерними є спади під час кризи 2008, 2011, 2015 та 2020 років, що є ключовими та дослідженими багатьма вченими. Динаміка індексу золота під час зазначених кризових подій також характеризувалися зростанням персистентності (кореляцій).

### 7.2.7 Обчислення фрактальної розмірності через нормалізовану щільність довжини

Це доволі простий показник, що відповідає середнім абсолютним послідовним різницям (стандартизованого) сигналу (np.mean(np.abs(np.diff(std\_signal)))). Цей метод було розроблено для вимірювання складності сигналів дуже короткої тривалості (< 30 відліків), і його можна використовувати, наприклад, коли цікавлять безперервні зміни фрактальної розмірності сигналу, обчислюючи його в межах ковзних вікон.

Для таких методів, як для фрактальної розмірності Хігучі, стандартне відхилення віконної фрактальної розмірності різко зростає, коли довжина ряду стає коротшою. Цей метод дає менше стандартне відхилення, особливо для коротших фрагментів, хоча і за рахунок меншої точності середнього значення фрактальної розмірності вікна.

Синтаксис процедури має наступний вигляд:

**fractal\_nld(signal, corrected=False)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **corrected** (*bool*) — якщо значення True, змінює масштаб вихідного значення фрактальної розмірності відповідно до степеневої моделі, щоб зробити його більш порівнянним з “істинним” значенням: FD = 1.9079\*((NLD-0.097178)^0.18383). Зауважте, що це може призвести до np.nan, якщо результат різниці буде від’ємним.

**Повертає**

* **NLDFD** (*float*) — фрактальна розмірність через нормалізовану щільність довжини.
* **info** (*dict*) — словник, що містить додаткову інформацію (наразі порожній, але повертається для узгодженості з іншими функціями).

#### 7.2.7.1 Віконна процедура

Для цього показника нам також не потребується нічого зайвого:

window = 250 # розмір вікна  
tstep = 1 # крок вікна  
ret\_type = 4 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням   
 # та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
nld\_corrected = True # нормалізація фрактальної розмірності  
  
length = len(time\_ser.values) # довжина ряду  
  
nldfd\_wind = [] # масив показників

Розпочинаємо віконну процедуру:

for i in tqdm(range(0,length-window,tstep)):  
   
 # відбираємо фрагменти  
 fragm = time\_ser.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо фрактальну розмірність   
 nld, \_ = nk.fractal\_nld(fragm,   
 corrected=nld\_corrected)  
  
 # зберігаємо результат до масиву значень  
 nldfd\_wind.append(nld)

100%|██████████| 5531/5531 [00:05<00:00, 994.20it/s]

Зберігаємо вихідні значення до текстового документа:

np.savetxt(f"fd\_nld\_name={symbol}\_wind={window}\_\  
 step={tstep}\_corrected={nld\_corrected}.txt", nldfd\_wind)

Оголошуємо мітки для рисунків та назви збережених рисунків:

label\_nld = fr'$NLDFD$'  
  
file\_name\_nld = f"fd\_nld\_name={symbol}\_wind={window}\_\  
 step={tstep}\_corrected={nld\_corrected}"

Виводимо результат:

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 nldfd\_wind,   
 ylabel,   
 label\_nld,  
 xlabel,  
 file\_name\_nld)

|  |
| --- |
| Рисунок 7.17: Динаміка індексу золота та фрактальної розмірності через нормалізовану щільність довжини |

[Рисунок 7.17](#fig-gold-wind-nld) показує, що спадає під час кризових та передкризових подій, що вказує на зростання кореляцій у дані періоди.

### 7.2.8 Обчислення фрактальної розмірності через нахил спектральної щільності потужності

Скористаємось наступним методом бібліотеки neurokit2:

**fractal\_psdslope(signal, method='voss1988', show=False, \*\*kwargs)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **method** (*str*) — метод оцінки фрактальної розмірності за нахилом, може бути "voss1988" (за замовчуванням) або "hasselman2013".
* **show** (*bool*) — якщо значення True, повертає графік залежності спектральної щільності потужності від частоти в логарифмічному масштабі.
* **kwargs** — інші аргументи, які слід передати до signal\_psd().

**Повертає**

* **slope** (*float*) — оцінка фрактальної розмірності, отримана в результаті аналізу нахилу спектральної щільності потужності.
* **info** (*dict*) — словник, що містить додаткову інформацію про параметри, які використовуються для аналізу нахилу спектральної щільності потужності.

#### 7.2.8.1 Для всього ряду

Для подальших розрахунків спочатку виконаємо перетворення ряду. Будемо використовувати вихідний часовий ряд для подальших розрахунків:

signal = time\_ser.copy()  
ret\_type = 1 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
for\_psd = transformation(signal, ret\_type)

І тепер виведемо графік залежності спектральної щільності потужності від частоти в логарифмічному масштабі:

psdslope, info = nk.fractal\_psdslope(for\_psd,  
 method="voss1988",  
 show=True)

|  |
| --- |
| Рисунок 7.18: Залежність спектральної щільності потужності від частоти в логарифмічному масштабі |

Як можна бачити з представленого графіку, спектральний нахил спектральної щільності потужності на різних частотах має лінійну залежність, а кут нахилу прямої, побудованої по спектру, близький до -2, що вказує на те, що динаміка індексу золота близька до дробового броунівського руху.

Тепер розглянемо варіацію кута нахилу спектра в рамках алгоритму ковзного вікна.

#### 7.2.8.2 Віконна процедура

Встановимо наступні параметри:

window = 250 # розмір вікна  
tstep = 1 # крок вікна  
ret\_type = 4 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням   
 # та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
method\_psd = "voss1988" # метод для розрахунку  
 # спектральної щільності  
  
length = len(time\_ser.values) # довжина ряду  
  
psd\_wind = [] # масив показників

Розпочинаємо віконну процедуру:

for i in tqdm(range(0,length-window,tstep)):  
   
 # відбираємо фрагменти  
 fragm = time\_ser.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо фрактальну розмірність   
 psd, \_ = nk.fractal\_psdslope(fragm, method=method\_psd)  
  
 # зберігаємо результат до масиву значень  
 psd\_wind.append(psd)

100%|██████████| 5531/5531 [00:16<00:00, 337.75it/s]

Зберігаємо вихідні значення до текстового документа:

np.savetxt(f"fd\_psd\_name={symbol}\_method{method\_psd}\_\  
 wind={window}\_step={tstep}.txt", psd\_wind)

Оголошуємо мітки для рисунків та назви збережених рисунків:

label\_psd = fr'$PSDFD$'  
  
file\_name\_psd = f"fd\_psd\_name={symbol}\_method{method\_psd}\_\  
 wind={window}\_step={tstep}"

Виводимо результат:

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 psd\_wind,   
 ylabel,   
 label\_psd,  
 xlabel,  
 file\_name\_psd)

|  |
| --- |
| Рисунок 7.19: Динаміка індексу золота та фрактальної розмірності через нахил спектральної щільності потужності |

Рисунок вище показує, що даний показник також реагує спадом у кризові та передкризові події, вказуючи на зростання автокореляції часового ряду під час даних подій. Також зрозуміло, що має місце варіації нахилу спектра щільності потужності. В одні моменти часу динаміка сигналу може бути подібна до броунівського руху, а в інші до білого шуму.

### 7.2.9 Обчислення кореляційної розмірності

Кореляційна розмірність (також позначається як ) є нижньою границею оцінки фрактальної розмірності досліджуваного фазового простору.

Спочатку здійснюється реконструкція фазового простору сигналі згідно методу часової затримки, і далі обчислюються відстані між усіма точками траєкторії. Потім обчислюється “кореляційна сума”, яка є часткою пар точок, відстань між якими менша за заданий радіус. Остаточна кореляційна розмірність апроксимується графіком залежності кореляційної суми від радіусу багатовимірного околу досліджуваних траєкторій, що будується в логарифмічному масштабі.

Цю розмірність можна викликати через fractal\_correlation(). Її синтаксис виглядає наступним чином:

**fractal\_correlation(signal, delay=1, dimension=2, radius=64, show=False, \*\*kwargs)**

**Параметри**

* **signal** (*Union[list, np.array, pd.Series]*) — сигнал (тобто часовий ряд) у вигляді вектора значень.
* **delay** (*int*) — часова затримка ().
* **dimension** (*int*) — розмірність вкладень ().
* **radius** (*Union[str, int, list]*) — послідовність радіусів для перевірки. Якщо передано ціле число, буде отримано експоненціальну послідовність довжиною заданим скалярним значенням radius у межах від 2.5% до 50% від діапазону відстані. Методи, реалізовані в інших пакетах, можна використовувати, вказуючи "nolds", "Corr\_Dim" або "boon2008".
* **show** (*bool*) — графік кореляційної розмірності, якщо True. За замовчуванням — False.
* **kwargs** — інші аргументи для передачі (наразі не використовуються).

**Повертає**

* **cd** (*float*) — кореляційна розмірність часового ряду.
* **info** (*dict*) — словник, що містить додаткову інформацію про параметри, які використовуються для обчислення кореляційної розмірності.

#### 7.2.9.1 Для всього часового ряду

Розглянемо залежність кореляційної суми від радіусу для всього часового ряду. Перш за все виконаємо перетворення ряду:

signal = time\_ser.copy()  
ret\_type = 1 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
for\_corr = transformation(signal, ret\_type)

Тепер розрахуємо кореляційну розмірність, побудувавши залежність кореляційної суми від радіусу в логарифмічному масштабі:

cd, info = nk.fractal\_correlation(for\_corr,  
 delay=1,   
 dimension=1,  
 radius="nolds",   
 show=True)

|  |
| --- |
| Рисунок 7.20: Залежність кореляційної суми від радіусу багатовимірного околу досліджуваних траєкторій. Графік побудований у логарифмічному масштабі |

Як ми можемо бачити, кореляційна сума дійсно має лінійну залежність для різних значень радіусу околу певної траєкторії, що вказує на фрактальність системи. Тепер подивимось як варіюється значення кореляційної розмірності в періоди турбулентності.

#### 7.2.9.2 Віконна процедура

Для цього показника визначимо наступні параметри:

window = 250 # розмір вікна  
tstep = 1 # крок вікна  
ret\_type = 1 # вид ряду:   
 # 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням   
 # та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
d\_wind = 2 # розмірність вкладень  
tau\_wind = 1 # часова затримка  
rad\_wind = "nolds" # метод для визначення масиву радіусів  
  
length = len(time\_ser.values) # довжина ряду  
  
corr\_wind = [] # масив показників

Розпочинаємо віконну процедуру:

for i in tqdm(range(0,length-window,tstep)):  
   
 # відбираємо фрагменти  
 fragm = time\_ser.iloc[i:i+window].copy()   
  
 # виконуємо процедуру трансформації ряду   
 fragm = transformation(fragm, ret\_type)  
  
 # розраховуємо фрактальну розмірність   
 cd\_wind, \_ = nk.fractal\_correlation(fragm,  
 delay=tau\_wind,   
 dimension=d\_wind,  
 radius=rad\_wind)  
  
 # зберігаємо результат до масиву значень  
 corr\_wind.append(cd\_wind)

100%|██████████| 5531/5531 [00:19<00:00, 290.43it/s]

Зберігаємо вихідні значення до текстового документа:

np.savetxt(f"fd\_correlation\_name={symbol}\_wind={window}\_\  
 step={tstep}\_dim={d\_wind}\_tau={tau\_wind}\_\  
 radius={rad\_wind}.txt", corr\_wind)

Оголошуємо мітки для рисунків та назви збережених рисунків:

label\_cd = fr'$CD$'  
  
file\_name\_cd = f"fd\_correlation\_name={symbol}\_wind={window}\_\  
 step={tstep}\_dim={d\_wind}\_tau={tau\_wind}\_\  
 radius={rad\_wind}"

Виводимо результат:

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 corr\_wind,   
 ylabel,   
 label\_cd,  
 xlabel,  
 file\_name\_cd)

|  |
| --- |
| Рисунок 7.21: Динаміка індексу золота та кореляційної фрактальної розмірності |

[Рисунок 7.21](#fig-gold-wind-cd) демонструє, що кореляційна розмірність для індексу золота також спадає у кризові та передкризові періоди, вказуючи на зростання корельованості теперішніх цін на золото з попередніми. Можна сказати і по іншому: у період криз трейдери починають самоорганізовуватись та колективно скупати або продавати відповідний актив; іншими словами, їх динаміка стає більш синхронною. Оскільки кореляційна розмірність вимірюється для траєкторій фазового простору, спад цього показника свідчить про зростання щільності досліджуваних траєкторій. Тобто, фазовий простір стає більш розрідженим, а всі його траєкторії концентрованими лише в одній конкретній області, що є індикатором згуртованості прихованих змінних досліджуваної системи.

# 8. Лабораторна робота № 7

**Тема.** Мультифрактальний аналіз складних систем.

**Мета.** Навчитися використовувати мультифрактальний аналіз детрендованих флуктуацій для отримання індикаторів нелінійних змін у системі.

## 8.1 Теоретичні відомості

### 8.1.1 Означення мультифракталів

У цій лабораторній ми викладемо основи теорії мультифракталів — **неоднорідних фрактальних об’єктів**, для повного опису яких, на відміну від регулярних фракталів, недостатньо введення лише однієї величини, його фрактальної розмірності , а потрібен цілий спектр таких розмірностей, кількість яких, взагалі кажучи, нескінченна. Причина цього полягає в тому, що поряд із суто геометричними характеристиками, які визначаються величиною , такі фрактали характеризуються й деякими специфічними статистичними властивостями.

|  |
| --- |
| Рисунок 8.1: Трикутник Серпинського, області якого згенеровані з нерівномірними ймовірностями |

Простіше всього пояснити, що розуміється під “неоднорідним фракталом” на прикладі трикутника Серпинського, отриманого за допомогою методу випадкових ітерацій.

Припустімо, однак, що в методі випадкових ітерацій ми тепер із якоїсь причини віддали перевагу одній із вершин трикутника, наприклад, вершині А, і стали вибирати її з імовірністю 90%. Дві ж інші вершини В і С для нас рівноцінні, але на їхню частку тепер припадає всього лише по 5%. Результат такої несиметричної гри” зображено нижче на рисунку вище.

Видно, що точки всередині трикутника АВС розподілені тепер вкрай нерівномірно. Більша їх частина перебуває біля вершини А та її прообразів. Водночас у вершин В і С (і їхніх прообразів) їх є вкрай мало. Проте, за звичайною термінологією, ця множина точок (за умови прагнення числа ітерацій до нескінченності) є фракталом, тому що збереглася основна властивість фракта — самоподібність. Дійсно, трикутник DFC, хоча в ньому у 20 разів менше точок, за своїми статистичними і геометричними властивостями повністю подібний до великого трикутника АВС. Так само, як і у великому трикутнику, точки в ньому концентруються здебільшого поблизу вершини D — аналогу вершини А.

|  |
| --- |
| Рисунок 8.2: Розподіл точок по трикутнику Серпинського, представленого на попередньому рисунку |

[Рисунок 8.2](#fig-dist-serp) більш детально демонструє результуючий розподіл точок по трикутнику Серпинського. Цифри в кожному з маленьких трикутників показують його відносну заселеність точками множини.

Однак, не дивлячись на нерівномірність розподілу точок фрактала, його фрактальна розмірність залишилась при цьому такою ж, . Покриття цієї множини все меншими трикутниками можна здійснити по тому ж алгоритму, що й раніше. Таке співпадіння змушує замислитись над пошуком нових кількісних характеристик, котрі могли б відрізнити нерівномірний розподіл точок від рівномірного.

Інший, складніший приклад неоднорідного фрактала, який ми б хотіли ще навести, показано на наступному рисунку. Ліворуч продемонстровано великий квадрат зі стороною, що дорівнює одиниці, який на цьому (нульовому) етапі повністю покриває собою деяку фрактальну множину точок . На наступному (першому) етапі, у центрі малюнка, показано, як ту саму множину можна покрити трьома меншими квадратами зі сторонами , у яких, відповідно, міститься частка та усіх точок.

|  |
| --- |
| Рисунок 8.3: Приклад мультифрактала, що підкоряється ренормалізаційній схемі |

Наступний етап покриття (зображений на рисунку праворуч) містить уже 9 квадратів зі сторонами (у нижньому правому куті) і (угорі праворуч і ліворуч). Відносна заселеність цих квадратів точками множини показана на рисунку. Вона відповідає добутку чинників заселеності (імовірностей): — для нижньої правої групи, — для верхньої лівої та — для верхньої правої групи. Зазначимо, що є чітка відповідність між заселеністю квадрата і його розмірами .

Подальший процес розбиття і покриття множини здійснюється згідно із цією ренормалізаційною схемою. Кожен квадрат, що має на -му кроці розмір і заселеність , замінюється на кроці трьома квадратами з розмірами і заселеностями відповідно, розміщеними таким самим чином відносно один одного, як показано на попередньому посередні рисунку.

Двоє розглянутих вище випадки являють собою приклади неоднорідних фракталів. Під словом “неоднорідний” ми тут розуміємо нерівномірний розподіл точок множини по фракталу або нерівномірний розподіл малих та великих флуктуацій у часовому ряді. Причина неоднорідності в попередніх випадках одна й та сама — різні ймовірності заповнення геометрично однакових елементів фрактала, або в загальному випадку невідповідність імовірностей заповнення геометричним розмірам відповідних областей. Такі неоднорідні фрактальні об’єкти в літературі називають мультифракталами, і їх вивченням ми й займемося надалі.

### 8.1.2 Узагальнені фрактальні розмірності

Дамо загальне визначення мультифрактала. Розглянемо фрактальний об’єкт, що займає якусь обмежену ділянку розміру у Евклідовому просторі з розмірністю . Нехай на якомусь етапі його побудови він являє собою множину з точок, якось розподілених у цій області. Ми будемо припускати, що врешті-решт . Прикладом такої множини може слугувати трикутник Серпінського, побудований методом випадкових ітерацій. Кожен крок ітераційної процедури додає до цієї множини одну нову точку.

Розіб’ємо всю область на кубічні клітинки зі стороною та об’ємом . Далі нас будуть цікавити тільки зайняті клітинки, у яких міститься хоча б одна точка. Нехай номер зайнятих комірок змінюється в межах , де — сумарна кількість зайнятих клітинок, яка, звісно, залежить від розміру клітинки .

Нехай представляє собою кількість точок у клітинці з номером , тоді величина

представляє собою ймовірність того, що навмання взята точка з нашої множини знаходиться в комірці . Інакше кажучи, ймовірності характеризують відносну заселеність комірок. З умови нормування ймовірності випливає, що

Уведемо тепер у розгляд **узагальнену статистичну суму** , що характеризується показником ступеня , який може набувати будь-яких значень в інтервалі

Спектр **узагальнених фрактальних розмірностей** , що характеризує даний розподіл точок в області , визначається за допомогою співвідношення

де функція має вид

Як ми покажемо нижче, якщо , тобто не залежить від , то дана множина точок являє собою звичайний, регулярний фрактал, який характеризується лише однією величиною — фрактальною розмірністю . Навпаки, якщо функція якось змінюється з , то розглянута множина точок представляє мультифрактал.

Таким чином, мультифрактал у загальному випадку характеризується деякою **нелінійною** функцією , що визначає поведінку статистичної суми при

Слід мати на увазі, що в реальній ситуації ми завжди маємо скінченне, хоча й дуже велике число дискретних точок , тому при комп’ютерному моделювані конкретної множини граничний перехід треба виконувати з обережністю, пам’ятаючи, що йому завжди передує ліміт .

Покажемо тепер, як поводиться узагальнена статистична сума у випадку звичайного регулярного фрактала з фрактальною розмірністю . У цьому випадку в усіх зайнятих комірках міститься однакова кількість точок

тобто фрактал представляється **однорідним**. Тоді очевидно, що відносні населеності клітинок, , також однакові, і узагальнена статистична сума набуває вигляду

Врахуємо тепер, що, згідно визначеню фрактальної розмірності , кількість зайнятих клітинок при достатньо малому поводить себе наступним чином:

Підставляючи (3) у формулу (2), і порівнюючи з (1), отримуємо

Ми приходимо до висновку, що у випадку звичайного фрактала функція (4) є лінійною. Тоді всі дійсно не залежать від . Фрактал у якого всі узагальнені фрактальні розмірності співпадають називається **монофракталом**.

Якщо розподіл точок по клітинкам неоднаковий, тоді фрактал називається неоднорідним, тобто представляє із себе мультифрактал, і для його характеристики необхідний цілий спектр узагальнених фрактальних розмірностей , кількість котрих, у загальному випадку, нескінченна.

Так, наприклад, при основний внесок в узагальнену статистичну суму (1) вносять комірки, що містять найбільшу кількість частинок у них і, відповідно, що характеризуються найбільшою ймовірністю їх заповнення . Навпаки, при основний внесок в узагальнену статистичну суму вносять найбільш розрідженні комірки з найменшою ймовірністю їх заповнення . Таким чином, функція показує, наскільки неоднорідним представляється досліджувана множина точок .

У подальшому для характеристики розподілу точок необхідно знати не тільки функцію , але і її похідну:

Ця похідна має важливий фізичний зміст, який буде продемонстровано пізніше. Зараз знову зазначимо, що для мультифрактальної системи вона не залишається константною і змінюється з .

### 8.1.3 Функція мультифрактального спектра

#### 8.1.3.1 Спектр фрактальних розмірностей

У попередньому пункті ми ввели поняття мультифрактала — об’єкта, що представляє собою неоднорідний фрактал. Для його опису ми ввели множину узагальнених фрактальних розмірностей , де приймає будь-які значення в інтервалі . Однак величини не є, строго кажучи, фрактальними розмірностями в загальному розумінні цього слова.

Тому часто поряд із ними для характеристики мультифрактальної множини використовують так звану **функцію мультифрактального спектра** (спектр сингулярностей мультифрактала), до якої, як ми побачимо надалі, більше підходить термін фрактальна розмірність. Ми покажемо, що величина фактично дорівнює хаусдорфовій розмірності якоїсь однорідної фрактальної підмножини із вихідної множини , що дає домінантний внесок у статистичну суму при заданій величині .

Однією з основних характеристик мультифрактала є набір імовірностей , що показують відносну заселеність клітинок , якими ми покриваємо цю множину. Чим менший розмір клітинки, тим менша величина її заселеності. Для самоподібних множин залежність від розміру клітинки має степеневий характер:

де являє собою деякий показник ступеня (різний для різнок клітинок ). Відомо, що для регулярного (однорідного) фрактала всі показники ступеня однакові й рівні фрактальній розмірності :

У даному випадку статистична сума (1) приймає наступний вигляд:

Тому і всі узагальнені фрактальні розмірності у цьому випадку співпадають та не залежать від .

Однак, для такого складного об’єкта, як мультифрактал, унаслідок його неоднорідності, ймовірності заповнення клітинок у загальному випадку різняться, і показник ступеня для різних клітинок може приймати різні значення. Достатньо типовою є ситуація, коли ці значення неперервно заповнюють деякий закритий інтервал , причому

Тепер перейдемо до питання о розподілі ймовірностей різних значень . Нехай є ймовірністю того, що знаходиться в інтервалі від до . Іншими словами, представляє собою відносну кількість клітинок , що характеризуються однією і тією самою мірою з , що лежать у цьому інтервалі. У випадку монофрактала, для котрого всі однакові (і рівні фрактальній розмірності ), це число, очевидно, пропорційно повній кількості клітинок , степеневим чином залежних від розміру клітинки . Показник ступеня в цьому співвідношені визначається фрактальною розмірністю множини .

Для мультифрактала, однак, це не так, і різні значення зустрічаються з ймовірністю, що характеризується не однією і тією ж величиною , а різними (в залежності від ) значеннями показниками ступеня ,

Таким чином, фізичний сенс функції полягає в тому, що вона представляє собою розмірність хаусдорфа деякої однорідної підмножини із вихідної множини , що характеризується однаковими ймовірностями заповнення клітинок . Оскільки фрактальна розмірність підмножини очевидно завжди менша або рівна фрактальній розмірності вихідної множини , має місце важлива нерівність для функції :

У результаті можна зробити висновок, що множина різних значень функції (при різних ) представляє собою **спектр фрактальних розмірностей** однорідних підмножин , на які можна розбити вихідну множину , кожна з яких характеризується власним значенням фрактальної розмірності .

Оскільки будь-якій підмножині належить лише частина загальної кількості клітинок , на котрі ми розділили вихідну множину , умова нормування ймовірностей, очевидно, не виконується при підсумовуванні тільки по цій підмножині. Сума цих імовірностей стає менше одиниці. Тому й самі ймовірності з одним і тим самим значенням очевидно менше (або в крайньому випадку одного порядку), ніж величина , яка обернено пропорційна кількості наявних клітинок, що покривають дану підмножину (нагадаємо, що у випадку монофрактала ). У результаті ми приходимо до наступної важливої нерівності для функції . А саме, при всіх значеннях

Знак рівності має місце, наприклад, для повністю однорідного фрактала, де .

#### 8.1.3.2 Перетворення Лежандра

Встановимо зв’язок функції із введенною раніше функцією . Обчислимо для цього статистичну суму . Підставляє у статистичну суму ймовірності , та переходячи від підсумовування по до інтегрування по з плотністю ймовірностей (5), ми отримаємо

Так як величина дуже мала, основний внесок у цей інтеграл вноситимуть ті значення , при яких показник ступеня виявляється мінімальним (а підінтегральна функція — максимальною). Цей вклад буде пропорційним значенню підінтегральної функції у точці максимума. Саме ж значення визначається при цьому з наступної умови:

Також, з умови мінімуму ми маємо

У результаті отримуємо, що залежність неявним чином визначається з рівняння

і що функція усюди є випуклою

Це значить, що величина дійсно визначає фрактальну розмірність підмножини , що має найбільший домінуючий внесок у статистичну суму (6) при заданій величині показника ступеня .

Оскільки , приходимо до висновку, що

Пам’ятаючи, що , можемо віднайти функцію :

Таким чином, якщо ми знаємо функцію мультифрактального спектра , то з домомогою співвідношень (7) та (8) ми можемо знайти функцію . Навпаки, знаючи , ми можемо знайти залежність з допомогою рівняння

і після цього знайти із (8) залежність . Ці два рівняння і визначають функцію .

Приймаючи до уваги те, що , і скорочуючи це рівняння на , приходимо до співвідношення

що еквівалентне виразу (9).

Вирази для та задають **перетворення Лежандра** від змінних до змінних :

та

Як відомо, для однорідного фрактала . Тому і . У цьому випадку “графік” функції на площині складається лише з однієї точки .

### 8.1.4 Мультифрактальний аналіз детрендованих флуктуацій

Монофрактальні та мультифрактальні структури фінансових сигналів є особливим різновидом масштабно-інваріантних структур. Найчастіше монофрактальна структура фінансових часових рядів визначається одним степеневим показником і передбачає, що масштабо-інваріантність не залежить від часу і простору. Однак, часто ми маємо змогу спостерігати просторово-часову варіацію масштабно-інваріантної структури досліджуваної складної системи. Ці просторово-часові варіації вказують на мультифрактальність фінансового сигналу, яка визначається мультифрактальним спектром. Ширина і форма мультифрактального спектра можуть також допомогти диференціювати варіативність серцевого ритму у пацієнтів із серцевими захворюваннями, такими як шлуночкова тахікардія, фібриляція шлуночків і застійна серцева недостатність. Мультифрактальний спектр може допомогти кількісно визначити асиметрію підйомів та спадів на фондовому чи криптовалютному ринках, передбачити фінансову кризу, що поступово наближується, і, таким чином, сприяти успішності подальших торгівельних рішень. Основна мета цього розділу — представити одну з найточніших процедур для визначення множини фрактальних показників — **мультифрактальний аналіз детрендованих флуктуацій** (multifractal detrended fluctuation analysis, MFDFA).

Побудова MFDFA складається з восьми кроків: Розділ *“Шум і випадкові блукання у часовому ряді”* представляє метод приведення часового ряду до такого, що подібний до випадкового блукання, що є попереднім кроком для MFDFA. Розділ *“Обчислення середньоквадратичної варіації часового ряду”* представляє середньоквадратичну варіацію, яка є основною процедурою для подальших обчислень в MFDFA і типовим способом обчислення середньої варіації часових рядів різної природи. Розділ *“Локальна середньоквадратична варіація часового ряду”* представляє обчислення локальної варіації часового ряду як середньоквадратичного відхилення часового ряду в межах сегментів, що можуть як перекриватися, так і не перекриватися. У розділі *“Локальне детрендування часового ряду”* таке ж локальне середньоквадратичне відхилення обчислюється навколо трендів, які часто зустрічаються у фінансових часових рядах. У розділі *“Монофрактальний аналіз детрендованих флуктуацій”* амплітуди локальних середніх квадратичних відхилень підсумовуються в узагальнене середнє квадратичне відхилення. У сумарному середньоквадратичному відхиленні для сегментів з малими розмірами вибірки переважають швидкі флуктуації часового ряду. На противагу цьому, у сумарному середньоквадратичному відхиленні для сегментів з великими розмірами вибірки переважають повільні коливання. Степенева залежність між загальним середнім квадратичним відхиленням для декількох розмірів вибірки (тобто масштабів) визначається за допомогою монофрактального аналізу дентрендованих флуктуацій (monofractal detrended fluctuation analysis, DFA) і називається показником Херста (Hurst exponent, ). У розділі *“Мультифрактальний аналіз детрендованих флуктуацій”* MFDFA отримують шляхом розширення на -й порядок узагальненого середньоквадратичного відхилення. Середньоквадратичне відхилення -го порядку може розрізняти сегменти з малими та великими флуктуаціями. Степенева залежність між середньоквадратичним відхиленням -го порядку чисельно визначається як показник Херста -го порядку. У розділі *“Мультифрактальний спектр часових рядів”* на основі показника Херста -го порядку обчислено декілька мультифрактальних спектрів.

Для подальшої візуалізації кожного кроку процедури MFDFA імпортуємо наступні модулі:

import matplotlib.pyplot as plt   
import numpy as np  
import neurokit2 as nk  
import yfinance as yf  
import pandas as pd  
import scienceplots  
from tqdm import tqdm  
  
import warnings  
warnings.filterwarnings('ignore')  
  
%matplotlib inline

І виконаємо налаштування рисунків для виведення:

plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься  
 # для виведення рисунків  
  
params = {  
 'figure.figsize': (8, 6), # встановлюємо ширину та висоту рисунків за замовчуванням  
 'font.size': 22, # розмір фонтів рисунку  
 'lines.linewidth': 2, # товщина ліній  
 'axes.titlesize': 'small', # розмір титулки над рисунком  
 "font.family": "sans-serif", # сімейство стилів підписів   
 "font.serif": ["Times"], # стиль підпису  
 'savefig.dpi': 300 # якість збережених зображень  
}  
  
plt.rcParams.update(params) # оновлення стилю згідно налаштувань

Цього разу розглянему можливість побудови індикаторів або індикаторів-передвісників на прикладі індексу сирої нафти West Texas Intermediate (WTI). При описі процедури MFDFA порівнюватимемо мультифрактальність даного ряду зі штучно зненерованими монофрактальними рядами складність яких має представлятися меншою у порівняні з нафтою.

Сам сайт Yahoo! Finance містить досить коротку історію щодених цін на нафту даної марки. Цього разу, в якості альтернативного прикладу, будемо послуговуватись альтернативним джерелом фінансових даних — [федеральним резервом економічних даних федерального резервного банку Сент-Луїса](https://fred.stlouisfed.org) (Federal Reserve Economic Data of the Federal Reserve Bank of St. Louis, FRED). На Python було створено бібліотеку для вивантаження даних із даного джерела — pandas-datareader. Для її встановлення достатньо прописати наступну команду:

!pip install pandas-datareader

Тепер імпортуємо відповідну бібліотеку:

import pandas\_datareader.data as web

та виконаємо зчитування індексу з FRED, використовуючи функціонал даної бібліотеки. Завантажимо дані за весь період, що нам буде доступний:

symbol = 'DCOILWTICO' # cимвол індексу, як указано на сайті FRED  
start = "1986-01-01" # Дата початку зчитування даних  
end = "2023-01-21" # Дата закінчення зчитування даних  
  
wti = web.DataReader(symbol, 'fred', start, end) # зчитуємо значення ряду   
time\_ser = wti[symbol].copy() # зберігаємо саме ціни закриття  
  
xlabel = 'time, days' # підпис по вісі Ох   
ylabel = symbol # підпис по вісі Оу

Виведемо досліджуваний ряд:

fig, ax = plt.subplots() # Створюємо порожній графік  
ax.plot(time\_ser.index, time\_ser.values) # Додаємо дані до графіку  
ax.legend([symbol]) # Додаємо легенду  
ax.set\_xlabel(xlabel) # Встановимо підпис по вісі Ох  
ax.set\_ylabel(ylabel) # Встановимо підпис по вісі Oy  
  
plt.xticks(rotation=45) # оберт позначок по осі Ох на 45 градусів  
  
plt.savefig(f'{symbol}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 8.4: Динаміка щоденних змін індексу сирої нафти WTI |

Розглянемо опис нашого масиву даних:

time\_ser.describe()

count 9336.000000  
mean 46.078743  
std 29.597897  
min -36.980000  
25% 19.990000  
50% 35.955000  
75% 67.242500  
max 145.310000  
Name: DCOILWTICO, dtype: float64

На представленому рисунку видно, що в значеннях досліджуваного індексу існують пропуски та негативні значення на ціну нафти. Для того, щоб позбутися від’ємного значення, можна виконати заміну значення на таке, що близьке до нуля. Для видалення значень NaN достатньо скористатися методом dropna() бібліотеки pandas.

time\_ser = time\_ser.dropna() # видаляємо значення NaN  
time\_ser[time\_ser.values<0] = 5 # замінюємо від'ємне значення на 5

Знову візуалізуємо результат:

fig, ax = plt.subplots() # Створюємо порожній графік  
ax.plot(time\_ser.index, time\_ser.values) # Додаємо дані до графіку  
ax.legend([symbol]) # Додаємо легенду  
ax.set\_xlabel(xlabel) # Встановимо підпис по вісі Ох  
ax.set\_ylabel(ylabel) # Встановимо підпис по вісі Oy  
  
plt.xticks(rotation=45) # оберт позначок по осі Ох на 45 градусів  
  
plt.savefig(f'{symbol}\_filtered.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 8.5: Динаміка щоденних змін індексу сирої нафти WTI (із видаленими NaN та заміненим від’ємним значенням) |

Останнє, що нам залишається, це приведення вихідного ряду до прибутковостей. Для цього визначимо функцію transformation() та знайдемо з її допомогою прибутковості:

def transformation(signal, ret\_type):  
  
 for\_rec = signal.copy()  
  
 if ret\_type == 1: # Зважаючи на вид ряду, виконуємо  
 # необхідні перетворення  
 pass  
 elif ret\_type == 2:  
 for\_rec = for\_rec.diff()  
 elif ret\_type == 3:  
 for\_rec = for\_rec.pct\_change()  
 elif ret\_type == 4:  
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 elif ret\_type == 5:   
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 for\_rec = for\_rec.abs()  
 elif ret\_type == 6:  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
  
 for\_rec = for\_rec.dropna().values  
  
 return for\_rec

signal = time\_ser.copy()  
ret\_type = 4 # вид ряду: 1 - вихідний,   
 # 2 - детрендований (різниця між теп. значенням та попереднім)  
 # 3 - прибутковості звичайні,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований ряд  
  
wti\_ret = transformation(signal, ret\_type) # знаходимо прибутковості   
wti\_length = len(wti\_ret) # визначаємо довжину прибутковостей

Як вже зазначалося, при описі процедури MFDFA, ми будемо послуговуватись для порівняння і монофрактальними сигналами. Для подальших розрахунків згенеруємо сигнал білого та рожевого шумів. У цьому нам може допомогти функція signal\_noise() бібліотеки neurokit2. Ця функція генерує чистий гаусовий (1/f)\*\*beta шум. Спектр потужності згенерованого шуму пропорційний S(f) = (1/f)\*\*beta. Було описано наступні категорії шуму:

* фіолетовий шум: beta = -2
* синій шум: beta = -1
* білий шум: beta = 0
* флікер / рожевий шум: beta = 1
* коричневий шум: beta = 2

Її синтаксис виглядає наступним чином:

**signal\_noise(duration=10, sampling\_rate=1000, beta=1, random\_state=None)**

**Параметри**

* **duration** (*float*) — бажана тривалість (у секундах).
* **sampling\_rate** (*int*) — бажана частота дискретизації (у Гц, тобто відліків на секунду).
* **beta** (*float*) — експонента шуму.
* **random\_state** (*None, int, numpy.random.RandomState або numpy.random.Generator*) — початкове значення (зерно) для генератора випадкових чисел.

**Повертає**

* **noise** (*array*) — сигнал чистого шуму.

Тепер можемо згенерувати 2 види шумів:

white\_noise = nk.signal\_noise(duration=wti\_length, # генеруємо білий шум   
 sampling\_rate=1,   
 beta=0,   
 random\_state=123)  
  
pink\_noise = nk.signal\_noise(duration=wti\_length, # генеруємо рожевий шум   
 sampling\_rate=1,   
 beta=1,   
 random\_state=123)

І тепер приступаємо до подальшої роботи.

#### 8.1.4.1 Шум і випадкові блукання у часовому ряді

Мультифрактальний аналіз детрендованих флуктуацій базується на класичному аналізі детрендованих флуктуацій (DFA). Класичний DFA застосовується до часових рядів зі структурою, подібною до випадкових блукань. Однак більшість фінансових часових рядів мають коливання, які більш схожі на прирости випадкових блукань. Якщо фінансовий часовий ряд має структуру, подібну до шуму, як у прибутковостей, його слід перетворити на випадково-блукаючий часовий ряд перед застосуванням DFA. Шуми можна перетворити на випадкові блукання шляхом віднімання середнього значення та інтегрування часового ряду (знаходження його кумулятивної суми). Часовий ряд з білим шумом, монофрактал (рожевий шум) і мультифрактал є шумовими часовими рядами і перетворюються на випадкові блукання за допомогою коду, що наведеного нижче:

RW1 = np.cumsum(white\_noise-np.mean(white\_noise)) # випадкове блукання білого шуму  
RW2 = np.cumsum(pink\_noise-np.mean(pink\_noise)) # випадкове блукання монофракталу  
RW3 = np.cumsum(wti\_ret-np.mean(wti\_ret)) # випадкове блукання нафти

fig, ax = plt.subplots(3, 1, figsize=(10, 8), sharex=True)  
  
ax[0].plot(time\_ser.index[1:], wti\_ret)  
ax[0].plot(time\_ser.index[1:], RW3, 'r')  
ax[0].grid(False)  
ax[0].margins(x=0)  
ax[0].set\_title('Мультифрактальний часовий ряд')  
  
ax[1].plot(time\_ser.index[1:], pink\_noise, label='Шумоподібний часовий ряд')  
ax[1].plot(time\_ser.index[1:], RW2, 'r', label='Випадкове блукання')  
ax[1].grid(False)  
ax[1].margins(x=0)  
ax[1].set\_title('Монофрактальний часовий ряд')  
ax[1].legend()  
  
ax[2].plot(time\_ser.index[1:], white\_noise)  
ax[2].plot(time\_ser.index[1:], RW1, 'r')  
ax[2].grid(False)  
ax[2].margins(x=0)  
ax[2].set\_title('Білий шум')  
  
plt.show();

|  |
| --- |
| Рисунок 8.6: Мультифрактальний (верхня панель), монофрактальний (середня панель) та подібний до білого шуму (нижня панель) часові ряди |

#### 8.1.4.2 Обчислення середньоквадратичної варіації часового ряду

Традиційний аналіз варіації часових рядів полягає в обчисленні середнього значення варіації як середньоквадратичного відхилення. Читач може скористатися наведеним нижче кодом для обчислення середньоквадратичного відхилення для часових рядів з білим шумом, монофрактальних і мультифрактальних даних:

RMS\_ordinary = np.sqrt(np.mean(white\_noise\*\*2)) # середньоквадратична варіація білого шуму  
RMS\_monofractal = np.sqrt(np.mean(pink\_noise\*\*2)) # середньоквадратична варіація монофрактала  
RMS\_multifractal = np.sqrt(np.mean(wti\_ret\*\*2)) # середньоквадратична варіація мультифрактала

fig, ax = plt.subplots(3, 1, figsize=(10, 8), sharex=True)  
  
ax[0].plot(time\_ser.index[1:], wti\_ret, label='Шумоподібний часовий ряд')  
ax[0].axhline(y=np.mean(wti\_ret), c='r', linestyle='--', label='Середнє')  
ax[0].axhline(y=np.mean(wti\_ret)+RMS\_multifractal, c='r', linestyle='-', label='+/- 1 RMS')  
ax[0].axhline(y=np.mean(wti\_ret)-RMS\_multifractal, c='r', linestyle='-')  
ax[0].set\_ylim(-20, 20)  
ax[0].grid(False)  
ax[0].margins(x=0)  
ax[0].set\_title('Мультифрактальний часовий ряд')  
  
ax[1].plot(time\_ser.index[1:], pink\_noise)  
ax[1].axhline(y=np.mean(pink\_noise), c='r', linestyle='--')  
ax[1].axhline(y=np.mean(pink\_noise)+RMS\_monofractal, c='r', linestyle='-')  
ax[1].axhline(y=np.mean(pink\_noise)-RMS\_monofractal, c='r', linestyle='-')  
ax[1].grid(False)  
ax[1].margins(x=0)  
ax[1].set\_title('Монофрактальний часовий ряд')  
  
ax[2].plot(time\_ser.index[1:], white\_noise)  
ax[2].axhline(y=np.mean(white\_noise), c='r', linestyle='--')  
ax[2].axhline(y=np.mean(white\_noise)+RMS\_ordinary, c='r', linestyle='-')  
ax[2].axhline(y=np.mean(white\_noise)-RMS\_ordinary, c='r', linestyle='-')  
ax[2].grid(False)  
ax[2].margins(x=0)  
ax[2].set\_title('Білий шум')  
  
handles, labels = ax[0].get\_legend\_handles\_labels()  
fig.legend(handles, labels, loc='lower center')  
  
plt.show();

|  |
| --- |
| Рисунок 8.7: Мультифрактальни (верхня панель), монофрактальний (середня панель) та подібний до білого шуму (нижня панель) часові ряди з нульовим середнім значенням (червона штрихова лінія) і RMS (червона суцільна лінія) |

[Рисунок 8.7](#fig-step-2) ілюструє, що середня амплітуда варіації (тобто середньоквадратичне відхилення) є однаковою для всіх трьох часових рядів — білого шуму, монофрактального та мультифрактального, навіть якщо вони мають досить різну структуру. MFDFA може розрізняти ці структури, як ми побачимо в наступних розділах.

#### 8.1.4.3 Локальна середньоквадратична варіація часового ряду

Мультифрактальні часові ряди на верхній панелі мають локальні флуктуації як з великими, так і з малими величинами. Середньоквадратичне відхилення (RMS) у попередньому коді можна обчислити для сегментів часового ряду, щоб розрізнити величини локальних флуктуацій. Проста процедура полягає в тому, щоб розділи часовий ряд на однакові за розміром сегменти, що не перекриваються, і обчислити локальне середнє квадратичне відхилення для кожного сегмента. Це можна зробити за допомогою коду наведеного нижче, і це є основною MFDFA:

def calc\_rms(arr, scale=1335, m=1):  
   
 # симулюємо випадкове блукання (X)  
 X = np.cumsum(arr - np.mean(arr))  
  
 # транспонуємо значення X   
 X = X.T   
  
 # визначаємо довжини сегментів   
 scale = scale  
   
 # визначаємо порядок полінома  
 m = m  
  
 # визначаємо кількість сегментів  
 segments = np.floor(len(X) / scale).astype(int)  
  
 Index = {} # словник індексів значень  
 fit = {} # словник для збереження отриманих поліноміальних кривих  
 # для кожного сегмента  
 RMS = [] # список середньоквадратичних відхилень  
  
 for v in range(segments+1): # проходимо по кожному сегменту  
 Idx\_start = v \* scale # визначаємо початкове значення сегмента   
 Idx\_stop = (v + 1) \* scale # визначаємо кінцеве значення  
   
 # формуємо масив індексів значень досліджуваного сегмента  
 Index[v] = np.arange(Idx\_start, min(Idx\_stop, len(X)))   
  
 # вилучаємо значення по індексам  
 X\_Idx = X[Index[v]]   
  
 # визначаємо поліноміальні коефіцієнти порядку m  
 C = np.polyfit(Index[v], X\_Idx, m)   
  
 # будуємо поліноміальну криву по визначеним коефіцієнтам  
 fit[v] = np.polyval(C, Index[v])   
   
 # визначаємо варіацію ряду навколо поліноміального тренда  
 RMS.append(np.sqrt(np.mean((X\_Idx - fit[v]) \*\* 2)))   
  
 return fit, RMS, Index, X

Перший рядок коду функції calc\_rms() перетворює зашумлений часовий ряд, мультифрактал, на часовий ряд випадкового блукання . Третій рядок коду задає масштаб параметра, який визначає розмір вибірки сегментів, що не перетинаються, для яких обчислюється локальне середнє квадратичне відхилення, RMS. П’ятий рядок — це кількість сегментів, на які можна розбити часовий ряд , де len(X) — розмір вибірки часового ряду . Таким чином, segments = 6 при len(X) = 9335 і scale = 1335. З дев’ятого по шістнадцятий рядок — це цикл, який обчислює локальне середньоквадратичне значення навколо тренду fit[v] для кожного сегмента, оновлюючи часовий індекс. У першому циклі v = 0, Index[0] переходить від 0 до 1335 значення сегмента (не включно). У другому циклі , Index[1] переходить від 1335 до 2670 значення другого сегмента. В останньому циклі , Index[6] переходить від 8010-го до 9345-го значення (не включно).

#### 8.1.4.4 Локальне детрендування часового ряду

У складних системах наявні повільні мінливі тренди, тому для кількісної оцінки масштабо-інваріантності флуктуацій навколо цих трендів необхідно провести детрендування сигналу. У попередньому коді до на кожному сегменті підбирається поліноміальний тренд fit[v]. Параметр визначає порядок полінома. Поліноміальний тренд є лінійним, якщо , квадратичним, якщо , і кубічним, якщо . Рядок C = np.polyfit(Index[v], X[Index[v]], m) визначає коефіцієнти полінома C, які використовуються для створення поліноміальної залежності тренду fit[v] для кожного сегмента. Потім для залишкової варіації, X(Index[v])-fit[v], обчислюється локальне середньоквадратичне відхилення, RMS[v], в межах кожного сегмента . Локальна середньоквадратична варіація, RMS[v], представлено на наступному рисунку як відстань між червоними пунктирними трендами і червоними суцільними лініями.

fit\_1, RMS\_1, Index\_1, X = calc\_rms(wti\_ret, scale=1335, m=1) # оцінка локального відхилення для мультифрактала  
fit\_2, RMS\_2, Index\_2, X = calc\_rms(wti\_ret, scale=1335, m=2) # оцінка локального відхилення для монофрактала  
fit\_3, RMS\_3, Index\_3, X = calc\_rms(wti\_ret, scale=1335, m=3) # оцінка локального відхилення для білого шуму

fig, ax = plt.subplots(3, 1, figsize=(10, 8), sharex=True)  
  
  
ax[0].plot(time\_ser.index[1:], X)  
for v in list(fit\_1.keys()):  
 ax[0].plot(time\_ser.index[Index\_1[v]], fit\_1[v], 'r--')  
 ax[0].plot(time\_ser.index[Index\_1[v]], fit\_1[v]+RMS\_1[v], c='r', linestyle='-')  
 ax[0].plot(time\_ser.index[Index\_1[v]], fit\_1[v]-RMS\_1[v], c='r', linestyle='-')  
  
ax[0].grid(False)  
ax[0].margins(x=0)  
ax[0].set\_title('Лінійне детрендування ' + r'$(m=1)$')  
  
  
  
ax[1].plot(time\_ser.index[1:], X, label='Випадкове блукання мультифрактального сигналу')  
for v in list(fit\_2.keys()):  
 if v == 1:  
 ax[1].plot(time\_ser.index[Index\_2[v]], fit\_2[v], 'r--', label='Локальний тренд')  
 ax[1].plot(time\_ser.index[Index\_2[v]], fit\_2[v]+RMS\_2[v], c='r', linestyle='-', label='+/- 1 RMS')  
 ax[1].plot(time\_ser.index[Index\_2[v]], fit\_2[v]-RMS\_2[v], c='r', linestyle='-')  
 else:  
 ax[1].plot(time\_ser.index[Index\_2[v]], fit\_2[v], 'r--')  
 ax[1].plot(time\_ser.index[Index\_2[v]], fit\_2[v]+RMS\_2[v], c='r', linestyle='-')  
 ax[1].plot(time\_ser.index[Index\_2[v]], fit\_2[v]-RMS\_2[v], c='r', linestyle='-')  
  
ax[1].grid(False)  
ax[1].margins(x=0)  
ax[1].set\_title('Квадратичне детрендування ' + r'$(m=2)$')  
  
  
  
ax[2].plot(time\_ser.index[1:], X)  
for v in list(fit\_3.keys()):  
 ax[2].plot(time\_ser.index[Index\_3[v]], fit\_3[v], 'r--')  
 ax[2].plot(time\_ser.index[Index\_3[v]], fit\_3[v]+RMS\_3[v], c='r', linestyle='-')  
 ax[2].plot(time\_ser.index[Index\_3[v]], fit\_3[v]-RMS\_3[v], c='r', linestyle='-')  
ax[2].grid(False)  
ax[2].margins(x=0)  
ax[2].set\_title('Кубічне детрендування ' + r'$(m=3)$')  
  
handles, labels = ax[1].get\_legend\_handles\_labels()  
fig.legend(handles, labels, loc='lower center')  
  
plt.show();

|  |
| --- |
| Рисунок 8.8: Обчислення локальних флуктуацій, RMS, навколо лінійного, квадратичного та кубічного трендів за допомогою функції calc\_rms() (, та , відповідно). Червона пунктирна лінія — це підігнаний тренд, fit[v], у шести сегментах вибірки розміром 1335. Відстань між червоним штриховим трендом і суцільними червоними лініями становить RMS. Локальні коливання, RMS, поблизу поліноміальних трендів є основою аналізу детрендованих флуктуацій |

#### 8.1.4.5 Монофрактальний аналіз детрендованих флуктуацій

У DFA варіації локального середньоквадратичного відхилення кількісно оцінюються загальним середньоквадратичним відхиленням ().

Швидкі коливання часового ряду впливатимуть на загальне середньоквадратичне відхилення, , у сегментах малої довжини (масштабу), тоді як повільні коливання впливатимуть на у сегментах великої довжини (масштабу). Таким чином, функція флуктуацій, , повинна бути обчислена для декількох масштабів, щоб виокремити вплив як швидкоплинних, так і повільних коливань, які у свою чергу визначають структурні перетворення часового ряду. Функція флуктуацій може бути обчислена для декількох масштабів шляхом модифікації попереднього коду:

def calc\_F(arr, scale, m=1):  
   
 X = np.cumsum(arr - np.mean(arr)) # симулюємо випадкове блукання (X)  
 X = X.T # транспонуємо значення X  
  
 scale = scale  
 m = m  
 segments = np.zeros(len(scale), dtype=int)  
 F = np.zeros(len(scale))  
  
 Index = {} # словник індексів значень  
 fit = {} # словник для збереження отриманих поліноміальних кривих  
 # для кожного сегмента  
 RMS = {} # словник середньоквадратичних відхилень  
  
 for ns in range(len(scale)):  
 segments[ns] = np.floor(len(X) / scale[ns]).astype(int)  
 RMS[ns] = np.zeros(segments[ns])  
  
 for v in range(segments[ns]): # проходимо по кожному сегменту  
 # визначаємо початкове значення сегмента  
 Idx\_start = v \* scale[ns]   
   
 # визначаємо кінцеве значення  
 Idx\_stop = (v + 1) \* scale[ns] if v < segments[ns] - 1 else len(X)   
   
 # формуємо масив індексів значень досліджуваного сегмента  
 Index[v, ns] = np.arange(Idx\_start, Idx\_stop)   
  
 # вилучаємо значення по індексам  
 X\_Idx = X[Index[v, ns]]   
  
 # визначаємо поліноміальні коефіцієнти порядку m  
 C = np.polyfit(Index[v, ns], X\_Idx, m)   
   
 # будуємо поліноміальну криву по визначеним коефіцієнтам  
 fit[v, ns] = np.polyval(C, Index[v, ns])   
  
 # оцінюємо середньоквадратичне відхилення для фрагмента v на масштабі ns   
 RMS[ns][v] = np.sqrt(np.mean((X\_Idx - fit[v, ns]) \*\* 2))   
  
 # оцінюємо загальне середньоквадратичне відхилення в межах масштабу ns  
 F[ns] = np.sqrt(np.mean(RMS[ns] \*\* 2))  
  
 return F, RMS, Index, X

scales = [16, 32, 64, 128, 256, 512, 1024][::-1]  
F, RMS, Index, X = calc\_F(wti\_ret, scale=scales) # оцінка узагальненої функції флуктуацій по різним масштабам

fig, ax = plt.subplots(len(scales), 1, figsize=(10, 8), sharex=True)  
  
for scale, val in enumerate(scales):  
 l = [Index[val] for val in Index.keys() if (val[1] == scale)]  
  
 x = np.array([])  
 for v in l:  
 x = np.concatenate([x, v])  
  
 y = np.array([])  
 for idx, v in enumerate(l):   
 y = np.concatenate([y, RMS[scale][idx]\*np.ones(len(v))])  
  
 if scales[scale] == 16:  
 ax[scale].plot(time\_ser.index[1:], y, c='b', label="Локальні флуктуацій: RMS")  
 ax[scale].axhline(y=F[scale], c='r', linestyle='-', label=r"RMS локальних флуктуацій: $F$")  
 ax[scale].grid(False)  
 ax[scale].set\_title(f"Масштаб = {scales[scale]}")  
 ax[scale].margins(x=0)  
 else:   
 ax[scale].plot(time\_ser.index[1:], y, c='b')  
 ax[scale].axhline(y=F[scale], c='r', linestyle='-')  
 ax[scale].grid(False)  
 ax[scale].set\_title(f"Масштаб = {scales[scale]}")  
 ax[scale].margins(x=0)   
  
handles, labels = ax[-1].get\_legend\_handles\_labels()  
fig.legend(handles, labels, loc='upper left', fontsize=12)  
  
fig.tight\_layout(pad=0.1)  
plt.show();

|  |
| --- |
| Рисунок 8.9: Локальні флуктуації, RMS[ns], обчислені для сегментів із різними масштабами. Функція флуктуацій F[ns] є загальним середньоквадратичним відхиленням локальних коливань RMS[ns]. Зверніть увагу, що F[ns] зменшується на менших масштабах |

DFA визначає монофрактальну структуру часового ряду відповідно до степеневої залежність між загальним середнім квадратичним відхиленням (тобто ), обчисленим для декількох масштабів. Степенева залежність між загальним середнім квадратичним відхиленням позначається нахилом () лінії регресії, розрахованим за допомогою коду, що представлений нижче:

C = np.polyfit(np.log(scales), np.log(F), 1)  
H = C[0]  
RegLine = np.polyval(C, np.log(scales))

Модифікуємо попередній код, додавши нові фрагменти:

def calc\_H(arr, scale, m=1):  
   
 X = np.cumsum(arr - np.mean(arr)) # симулюємо випадкове блукання (X)  
 X = X.T # транспонуємо значення X  
  
 scale = scale  
 m = m  
 segments = np.zeros(len(scale), dtype=int)  
 F = np.zeros(len(scale))  
  
 Index = {} # словник індексів значень  
 fit = {} # словник для збереження отриманих поліноміальних кривих  
 # для кожного сегмента  
 RMS = {} # словник середньоквадратичних відхилень  
  
 for ns in range(len(scale)):  
 segments[ns] = np.floor(len(X) / scale[ns]).astype(int)  
 RMS[ns] = np.zeros(segments[ns])  
  
 for v in range(segments[ns]): # проходимо по кожному сегменту  
 # визначаємо початкове значення сегмента  
 Idx\_start = v \* scale[ns]   
   
 # визначаємо кінцеве значення  
 Idx\_stop = (v + 1) \* scale[ns] if v < segments[ns] - 1 else len(X)   
   
 # формуємо масив індексів значень досліджуваного сегмента  
 Index[v, ns] = np.arange(Idx\_start, Idx\_stop)   
  
 # вилучаємо значення по індексам  
 X\_Idx = X[Index[v, ns]]   
  
 # визначаємо поліноміальні коефіцієнти порядку m  
 C = np.polyfit(Index[v, ns], X\_Idx, m)   
   
 # будуємо поліноміальну криву по визначеним коефіцієнтам  
 fit[v, ns] = np.polyval(C, Index[v, ns])   
  
 # оцінюємо середньоквадратичне відхилення для фрагмента v на масштабі ns   
 RMS[ns][v] = np.sqrt(np.mean((X\_Idx - fit[v, ns]) \*\* 2))   
  
 # оцінюємо загальне середньоквадратичне відхилення в межах масштабу ns  
 F[ns] = np.sqrt(np.mean(RMS[ns] \*\* 2))  
  
 # знаходимо коефіцієнти рівняння прямої   
 C = np.polyfit(np.log(scale), np.log(F), 1)   
   
 # беремо кут нахилу прямої в якості показника Херста  
 H = C[0]  
  
 # будуємо саме рівняння  
 RegLine = np.polyval(C, np.log(scale))  
  
 return H, RegLine, F

Тепер розглянемо залежність загальної функції флуктуацій від різних довжин (масштабів) локальних сегментів ряду для досліджуваних нами рядів:

scmin = 16  
scmax = 1024  
scres = 19  
exponents = np.linspace(np.log(scmin), np.log(scmax), scres)  
  
scales\_exp = np.round(np.exp(1)\*\*exponents).astype(int)  
  
H\_multifrac, RegLine\_multifrac, F\_multifrac = calc\_H(wti\_ret, scale=scales\_exp, m=1)  
H\_monofrac, RegLine\_monofrac, F\_monofrac = calc\_H(pink\_noise, scale=scales\_exp, m=1)  
H\_white\_noise, RegLine\_white\_noise, F\_white\_noise = calc\_H(white\_noise, scale=scales\_exp, m=1)

fig, ax = plt.subplots(1, 1)  
  
ax.set\_xscale('log')  
ax.set\_yscale('log')  
ax.scatter(scales\_exp, F\_multifrac,   
 label=fr"Мультифрактальний ряд ($H$={H\_multifrac:.2f})",   
 color='darkblue')  
plt.plot(scales\_exp, np.exp(RegLine\_multifrac), color='darkblue')  
  
ax.scatter(scales\_exp, F\_monofrac,   
 label=fr"Монофрактальний ряд ($H$={H\_monofrac:.2f})",   
 color='magenta')  
plt.plot(scales\_exp, np.exp(RegLine\_monofrac), color='magenta')  
  
  
ax.scatter(scales\_exp, F\_white\_noise,   
 label=fr"Білий шум ($H$={H\_white\_noise:.2f})",   
 color='red')  
plt.plot(scales\_exp, np.exp(RegLine\_white\_noise), color='red')  
  
ax.set\_xlabel(r'$\log{ns}$')  
ax.set\_ylabel(r"$\log{F(ns)}$")  
  
plt.legend(fontsize=12)  
  
fig.tight\_layout()  
plt.show();

|  |
| --- |
| Рисунок 8.10: Графік залежності загального середньоквадратичного відхилення (тобто функції флуктуацій ) від масштабу, де і , і масштаб представлені в логарифмічних координатах. Масштабно-інваріантна залежність позначається нахилом, , ліній регресії. Нахил, , є показником степеневого закону, який називається показником Херста, оскільки і масштаб представлені в логарифмічних координатах |

Як уже зазначалося в попередній лабораторній роботі, кут нахилу ліній регресії називається показником Херста. Показник Херста визначає монофрактальну структуру часового ряду, показуючи, наскільки швидко зростає загальне середньоквадратичне відхилення, , локальних коливань, RMS, зі збільшенням розміру локальних сегментів ряду (тобто, масштабу). [Рисунок 8.10](#fig-step-6) показує, що загальне середньоквадратичне значення локальних флуктуацій, , зростає швидше зі збільшенням розміру вибірки сегментів для монофрактального рожевого шуму порівняно з індексом нафти та білим шумом. Більший показник Херста, , візуально свідчить про повільнішу еволюцію варіацій (тобто більш стійку структуру) в рожевому шумі порівняно з нафтою та білим шумом. [Рисунок 8.11](#fig-hurst) ілюструє, що показник Херста визначає континуум між часовими рядами, подібними до шуму, і часовими рядами, подібними до випадкового блукання. Показник Херста знаходиться в інтервалі від 0 до 1 для часових рядів, подібних до шуму, тоді як для часових рядів, подібних до випадкових блукань, він перевищує 1. Часовий ряд має довгострокову залежну (тобто корельовану) структуру, коли показник Херста знаходиться в інтервалі 0.5-1, і антикорельовану структуру, коли показник Херста знаходиться в інтервалі 0-0.5. Часовий ряд має незалежну або короткострокову залежну структуру в окремому випадку, коли показник Херста дорівнює 0.5. Згідно з попереднім рисунком, часові ряди білого шуму та нафти представляються непередбачуваними, оскільки показник Херста близький до 0.5, тоді як рожевий шум довгостроково залежну структуру з показником Херста близьким до 1.

betas = np.linspace(0.0, 2.0, 12)[::-1]  
scmin = 16  
scmax = 1024  
scres = 19  
exponents = np.linspace(np.log(scmin), np.log(scmax), scres)  
scales\_exp = np.round(np.exp(1)\*\*exponents).astype(int)

color = iter(plt.cm.rainbow(np.linspace(0, 1, len(betas))))  
  
fig, ax = plt.subplots(len(betas), 1, figsize=(10, 8), sharex=True)  
  
for idx, beta in enumerate(betas):  
  
 noise = nk.signal\_noise(duration=wti\_length, # генеруємо шум із різними значеннями beta   
 sampling\_rate=1,   
 beta=beta,   
 random\_state=123)   
  
 H\_noise, \_, \_ = calc\_H(arr=noise, scale=scales\_exp, m=1)  
  
 c = next(color)  
 ax[idx].plot(np.arange(len(noise)), noise, label=fr"$H$ = {H\_noise:.2f}", c=c)  
 ax[idx].grid(False)  
 ax[idx].legend(loc="upper left")  
 ax[idx].margins(x=0)  
  
fig.subplots\_adjust(hspace=0)  
  
plt.show();

|  |
| --- |
| Рисунок 8.11: Діапазон показників Херста визначає континуум фрактальних структур між білим шумом () і коричневим шумом (). Рожевий шум розділяє шуми , які мають більш помітні швидкі флуктуації, і випадкові блукання , які мають більш помітні повільні флуктуації |

#### 8.1.4.6 Мультифрактальний аналіз детрендованих флуктуацій

Структури монофрактального та мультифрактального часових рядів відрізняється, хоча вони мають схожі загальні середньоквадратичні значення. Мультифрактальні часові ряди мають локальні флуктуації як з екстремально малими, так і з екстремально великими значеннями, що не характерно монофрактальним часовим рядам. Відсутність флуктуацій з екстремально великими та малими значеннями призводить до нормального розподілу для монофрактального часового ряду, де варіація описується лише статистичним моментом другого порядку (тобто дисперсією). Отже, монофрактальний DFA базується на статистиці другого порядку загального середньоквадратичного відхилення (тобто, ). У мультифрактальному часовому ряді локальні коливання, RMS[ns][v], будуть екстремально великими для сегментів в межах часових періодів великих коливань і екстремально малими для сегментів в межах часових періодів малих коливань. Отже, мультифрактальні часові ряди не є нормально розподіленими і слід враховувати всі статистичні моменти -го порядку. Таким чином, необхідно розширити загальне середньоквадратичне значення монофрактального DFA (тобто ) до середньоквадратичної функції флуктуацій -го порядку мультифрактального DFA ():

def calc\_Fq(arr, scale, q, m=1):  
   
 X = np.cumsum(arr - np.mean(arr)) # симулюємо випадкове блукання (X)  
 X = X.T # транспонуємо значення X  
  
 scale = scale   
 qs = q  
 m = m  
 segments = np.zeros(len(scale), dtype=int)  
 Fq = np.zeros((len(qs), len(scale)))  
 Index = {}  
 RMS = {} # словник локальних середньоквадратичних відхилень  
 fit = {} # словник для збереження отриманих поліноміальних кривих  
 # для кожного сегмента  
 qRMS = {} # словник локальних відхилень зважених показником q  
  
 for ns in range(len(scale)):  
 segments[ns] = np.floor(len(X) / scale[ns]).astype(int)  
 RMS[ns] = np.zeros(segments[ns])  
  
 # проходимо по кожному сегменту  
 for v in range(segments[ns]):   
  
 # визначаємо початкове значення сегмента  
 Idx\_start = v \* scale[ns]   
   
 # визначаємо кінцеве значення  
 Idx\_stop = (v + 1) \* scale[ns] if v < segments[ns] - 1 else len(X)   
   
 # формуємо масив індексів значень досліджуваного сегмента  
 Index[v] = np.arange(Idx\_start, Idx\_stop)   
  
 # вилучаємо значення по індексам  
 X\_Idx = X[Index[v]]   
  
 # визначаємо поліноміальні коефіцієнти порядку m  
 C = np.polyfit(Index[v], X\_Idx, m)   
   
 # будуємо поліноміальну криву по визначеним коефіцієнтам  
 fit = np.polyval(C, Index[v])   
  
 # оцінюємо середньоквадратичне відхилення для фрагмента v на масштабі ns   
 RMS[ns][v] = np.sqrt(np.mean((X\_Idx - fit) \*\* 2))   
   
 # приводимо q значення до типу float  
 qs = np.asarray\_chkfinite(qs, dtype=float)  
  
 # для мультифрактальності  
 # ----------------------------  
 for nq, qval in enumerate(qs):  
 if (qval != 0.):   
 qRMS[nq, ns] = RMS[ns] \*\* q[nq]  
 Fq[nq, ns] = np.mean(qRMS[nq, ns]) \*\* (1 / q[nq])  
 else:  
 Fq[nq, ns] = np.exp(0.5 \* np.mean(np.log(RMS[ns] \*\* 2)))  
 # ----------------------------  
  
 return Fq, qRMS, Index

У новому блоці коду запускається цикл, який обчислює загальне середньоквадратичне значення -порядку, , від від’ємних до додатних . Порядок зважує вплив сегментів ряду з великими та малими коливаннями, RMS, як показано на наступному рисунку. На для від’ємних впливають сегменти з малими RMS(v). Навпаки, на для додатних впливають відрізки з великими RMS(v). Локальні флуктуації RMS з великими та малими величинами класифікуються за величиною від’ємного або додатного порядку відповідно. На для і більше впливають відрізки з найменшим і найбільшим RMS(v), відповідно, порівняно з для і . Середня точка є нейтральною до впливу відрізків з малим та великим RMS. Зверніть увагу, що в останньому рядку коду нового блоку перевизначено окремий випадок , оскільки прямує до нескінченності (тобто, ). Читач також повинен помітити, що дорівнює статистиці другого порядку , оскільки . Монофрактальний DFA тепер розширюється до MFDFA.

scales = np.array([32])  
nq = np.array([-3, -1, 1, 3])  
  
Fq, qRMS, Index = calc\_Fq(wti\_ret, scale=scales, q=nq, m=1)  
Fq\_pink, qRMS\_pink, Index = calc\_Fq(pink\_noise, scale=scales, q=nq, m=1)

fig, ax = plt.subplots((len(nq)+1), 1, figsize=(12, 10))  
  
ax[0].plot(time\_ser.index[1:], wti\_ret, label="Мультифрактал")  
ax[0].plot(time\_ser.index[1:], pink\_noise, label="Монофрактал")  
ax[0].grid(False)  
ax[0].margins(x=0)  
ax[0].legend(loc='upper left', fontsize=12)  
ax[0].get\_xaxis().set\_visible(False)  
  
  
for idx in range(1, len(nq)+1):  
 l = [Index[val] for val in Index.keys()]  
  
 x = np.array([])  
 for v in l:  
 x = np.concatenate([x, v])  
  
 y = np.array([])  
 y\_pink = np.array([])  
 for i, v in enumerate(l):   
 y = np.concatenate([y, qRMS[(idx-1, 0)][i]\*np.ones(len(v))])  
 y\_pink = np.concatenate([y\_pink, qRMS\_pink[(idx-1, 0)][i]\*np.ones(len(v))])  
   
 ax[idx].set\_title(fr"Локальні варіації для {scales[0]}-го масштабу при $q=${nq[idx-1]}", fontsize=12)  
 ax[idx].plot(time\_ser.index[1:], y)  
 ax[idx].plot(time\_ser.index[1:], y\_pink)  
 ax[idx].grid(False)  
 ax[idx].margins(x=0)   
   
 if idx != len(nq):   
 ax[idx].get\_xaxis().set\_visible(False)  
  
handles, labels = ax[0].get\_legend\_handles\_labels()  
  
plt.show();

|  |
| --- |
| Рисунок 8.12: Ілюстрація залежності локальних флуктуацій qRMS від при масштабі 32. qRMS — це -порядок локальних флуктуацій (тобто, RMS) і є складовою частиною загального -порядку RMS (тобто, $F\_{q}). qRMS представлено для монофрактального (зелена смуга) та мультифрактальних (синя смуга) часових рядів. Від’ємний порядок ( і ) підсилює сегменти в мультифрактальному часовому ряді з екстремально малими RMS, тоді як додатний порядок ( і ) підсилює відрізки з екстремально великими RMS. Зверніть увагу, що і підсилюють малу і велику варіацію відповідно більше, ніж і . Зауважте також, що монофрактальний часовий ряд не має відрізків з екстремально великими або малими коливаннями і, таким чином, не має піків у qRMS. Загальне середньоквадратичне відхилення -го порядку здатне розрізняти структуру малих і великих флуктуацій і, відповідно, монофрактальних і мультифрактальних часових рядів |

Тепер можна визначити показники Херста -го порядку як нахили () ліній регресії для кожного середньоквадратичного значення -го порядку (). І , і лінія регресії визначаються в циклі для кожного -го порядку:

def calc\_Hq(arr, scale, q, m=1):  
   
 X = np.cumsum(arr - np.mean(arr)) # симулюємо випадкове блукання (X)  
 X = X.T # транспонуємо значення X  
  
 scale = scale   
 qs = q  
 m = m  
 segments = np.zeros(len(scale), dtype=int)   
 Fq = np.zeros((len(qs), len(scale))) # масив для збереження загальної функції флуктуацій   
 hq = np.zeros(len(qs), dtype=float) # масив для збереження Херста q-го порядку  
 qRegLine = {} # словник для збереження ліній регресій  
 Index = {} # словник для збереження індексів сегментів ряду  
 RMS = {} # словник локальних середньоквадратичних відхилень  
 fit = {} # словник для збереження отриманих поліноміальних кривих  
 # для кожного сегмента  
 qRMS = {} # словник локальних відхилень зважених показником q  
  
 for ns in range(len(scale)):  
 segments[ns] = np.floor(len(X) / scale[ns]).astype(int)  
 RMS[ns] = np.zeros(segments[ns])  
  
 # проходимо по кожному сегменту  
 for v in range(segments[ns]):   
  
 # визначаємо початкове значення сегмента  
 Idx\_start = v \* scale[ns]   
   
 # визначаємо кінцеве значення  
 Idx\_stop = (v + 1) \* scale[ns] if v < segments[ns] - 1 else len(X)   
   
 # формуємо масив індексів значень досліджуваного сегмента  
 Index[v] = np.arange(Idx\_start, Idx\_stop)   
  
 # вилучаємо значення по індексам  
 X\_Idx = X[Index[v]]   
  
 # визначаємо поліноміальні коефіцієнти порядку m  
 C = np.polyfit(Index[v], X\_Idx, m)   
   
 # будуємо поліноміальну криву по визначеним коефіцієнтам  
 fit = np.polyval(C, Index[v])   
  
 # оцінюємо середньоквадратичне відхилення для фрагмента v на масштабі ns   
 RMS[ns][v] = np.sqrt(np.mean((X\_Idx - fit) \*\* 2))   
   
 # приводимо q значення до типу float  
 qs = np.asarray\_chkfinite(qs, dtype=float)  
  
 # для мультифрактальності  
 # ----------------------------  
 for nq, qval in enumerate(qs):  
 if (qval != 0.):   
 qRMS[nq, ns] = RMS[ns] \*\* q[nq]  
 Fq[nq, ns] = np.mean(qRMS[nq, ns]) \*\* (1 / q[nq])  
 else:  
 Fq[nq, ns] = np.exp(0.5 \* np.mean(np.log(RMS[ns] \*\* 2)))  
  
 for nq, \_ in enumerate(qs):   
 C = np.polyfit(np.log(scale), np.log(Fq[nq, :]), m)  
 hq[nq] = C[0]  
 qRegLine[nq] = np.polyval(C, np.log(scale))  
 # ----------------------------  
  
 return hq, qRegLine, Fq

scmin = 16  
scmax = 1024  
scres = 19  
  
nq = np.linspace(-5, 5, 11)  
  
exponents = np.linspace(np.log(scmin), np.log(scmax), scres)  
scales\_exp = np.round(np.exp(1)\*\*exponents).astype(int)  
  
Hq\_multifrac, qRegLine\_multifrac, Fq\_multifrac = calc\_Hq(wti\_ret, scale=scales\_exp, q=nq, m=1)  
Hq\_monofrac, qRegLine\_monofrac, Fq\_monofrac = calc\_Hq(pink\_noise, scale=scales\_exp, q=nq, m=1)  
Hq\_white\_noise, qRegLine\_white\_noise, Fq\_white\_noise = calc\_Hq(white\_noise, scale=scales\_exp, q=nq, m=1)

fig, ax = plt.subplots(2, 2, figsize=(15, 10))  
  
ax[0][0].set\_title("Мультифрактал")  
ax[0][0].set\_xlabel(r"$ns$")  
ax[0][0].set\_ylabel(r"$F\_{q}(ns)$")  
ax[0][0].set\_xscale('log')  
ax[0][0].set\_yscale('log')  
for i in range(len(nq)):  
 ax[0][0].scatter(scales\_exp, Fq\_multifrac[i, :], color='darkblue')  
 ax[0][0].plot(scales\_exp, np.exp(qRegLine\_multifrac[i]), color='darkblue')  
  
ax[0][1].set\_title("Монофрактал")  
ax[0][1].set\_xlabel(r"$ns$")  
ax[0][1].set\_xscale('log')  
ax[0][1].set\_yscale('log')  
for i in range(len(nq)):  
 ax[0][1].scatter(scales\_exp, Fq\_monofrac[i, :], color='magenta')  
 ax[0][1].plot(scales\_exp, np.exp(qRegLine\_monofrac[i]), color='magenta')  
  
ax[1][0].set\_title("Білий шум")  
ax[1][0].set\_xlabel(r"$ns$")  
ax[1][0].set\_ylabel(r"$F\_{q}(ns)$")  
ax[1][0].set\_xscale('log')  
ax[1][0].set\_yscale('log')  
for i in range(len(nq)):  
 ax[1][0].scatter(scales\_exp, Fq\_white\_noise[i, :], color='red')  
 ax[1][0].plot(scales\_exp, np.exp(qRegLine\_white\_noise[i]), color='red')  
  
ax[1][1].set\_title(r"Показники Херста $q$-го порядку")  
ax[1][1].set\_xlabel(r"$q$")  
ax[1][1].set\_ylabel(r"$h(q)$")  
ax[1][1].plot(nq, Hq\_multifrac, linestyle='-', marker='o', label="Мультифрактал", color='darkblue')  
ax[1][1].plot(nq, Hq\_monofrac,linestyle='-', marker='o', label="Монофрактал", color='magenta')  
ax[1][1].plot(nq, Hq\_white\_noise, linestyle='-', marker='o', label="Білий шум", color='red')  
ax[1][1].legend()  
  
fig.tight\_layout()  
plt.show();

|  |
| --- |
| Рисунок 8.13: Середньоквадратичні значення для різних -их порядків та відповідні лінії регресії обчислені за допомогою MFDFA для мультифракталу, монофракталу та білого шуму |

Можемо бачити, що узагальнена функція флуктуацій для мультифракталу залежить не лише від масштабу, але й від , що демонструють різні нахили ліній регресії . Масштабуючі узагальнені функції флуктуацій для монофракталу та білого шуму є -незалежними, оскільки їх лінії регресії для різних масштабів мають один і той самий кут нахилу. Показник Херста -го порядку для мультифрактального (синя лінія) ряду представляється незалежним для і змінним для . Це вказує на те, що джерелом мультифрактальності нафти є аномально великі флуктуацій як, наприклад, криза коронавірусної пандемії. Для монофракталу (рожева лінія) та білого шуму (червона лінія) залишаються сталими.

#### 8.1.4.7 Мультифрактальний спектр часових рядів

Показник Херста -го порядку є лише одним з декількох типів масштабних показників, що використовуються для параметризації мультифрактальної структури часових рядів. Як уже було представлено попередньо, ми можемо вивести показник маси -го порядку (), а потім через отримати показник сингулярності -го порядку () і фрактальну розмірність () флуктуацій (областей) із ступенем сингулярності . Графік залежності від представляє мультифрактальний спектр. Показники маси, сингулярності та фрактальності можна обчислити згідно коду, що наведений нижче:

tau\_multifrac = nq \* Hq\_multifrac - 1   
tau\_monofrac = nq \* Hq\_monofrac - 1   
tau\_white\_noise = nq \* Hq\_white\_noise - 1   
  
alpha\_multifrac = np.gradient(tau\_multifrac, nq)  
alpha\_monofrac = np.gradient(tau\_monofrac, nq)  
alpha\_white\_noise = np.gradient(tau\_white\_noise, nq)  
  
f\_multifrac = nq \* alpha\_multifrac - tau\_multifrac  
f\_monofrac = nq \* alpha\_monofrac - tau\_monofrac  
f\_white\_noise = nq \* alpha\_white\_noise - tau\_white\_noise

fig, ax = plt.subplots(1, 3, figsize=(12, 8))  
  
ax[0].set\_xlabel(r"$q$")  
ax[0].set\_ylabel(r"$\tau(q)$")  
ax[0].plot(nq, tau\_multifrac, linestyle='-', marker='o', label="Мультифрактал", color='darkblue')  
ax[0].plot(nq, tau\_monofrac, linestyle='-', marker='o', label="Монофрактал", color='magenta')  
ax[0].plot(nq, tau\_white\_noise, linestyle='-', marker='o', label="Білий шум", color='red')  
ax[0].legend()  
  
ax[1].set\_xlabel(r"$\alpha$")  
ax[1].set\_ylabel(r"$f(\alpha)$")  
ax[1].plot(alpha\_multifrac, f\_multifrac, linestyle='-', marker='o', label="Мультифрактал", color='darkblue')  
ax[1].plot(alpha\_monofrac, f\_monofrac, linestyle='-', marker='o', label="Монофрактал", color='magenta')  
ax[1].plot(alpha\_white\_noise, f\_white\_noise, linestyle='-', marker='o', label="Білий шум", color='red')  
  
ax[2].set\_xlabel(r"$q$")  
ax[2].set\_ylabel(r"$f(\alpha)$")  
ax[2].plot(nq, f\_multifrac, linestyle='-', marker='o', label="Мультифрактал", color='darkblue')  
ax[2].plot(nq, f\_monofrac, linestyle='-', marker='o', label="Монофрактал", color='magenta')  
ax[2].plot(nq, f\_white\_noise, linestyle='-', marker='o', label="Білий шум", color='red')  
  
fig.tight\_layout(pad=0.3)   
plt.show();

|  |
| --- |
| Рисунок 8.14: Множинне представляння мультифрактального спектру для мультифрактала, монофрактала та білого шуму |

# 9. Лабораторна робота № 11

**Тема.** Аналіз незворотності часового ряду.

**Мета.** Навчитись розраховувати значення індексів незворотності часу для складних сигналів та досліджувати їх динаміку у випадку незворотних змін, побудови передвісників критичних і кризових явищ.

## 9.1 Теоретичні відомості

Незворотність часу є фундаментальною властивістю нерівноважних дисипативних систем, і її втрата може свідчити про розвиток деструктивних процесів.

З огляду на статистичні властивості досліджуваного сигналу, його еволюцію можна було б назвати незворотною, якби була відсутня інваріантність, тобто був би отриманий той же сигнал, якби ми виміряли його в протилежному напрямку. Функція може бути застосована для знаходження характеристик, які відрізняються прямою і зворотною версіями, тобто часові ряди незворотні, якщо . Основна ідея цього визначення полягає в тому, що немає ніяких обмежень на .

Передбачається, що стаціонарний процес називається статистично зворотним у часі, якщо розподіл ймовірностей прямої та зворотної систем приблизно однаковий. Незворотність часових рядів вказує на наявність нелінійних залежностей (пам’яті) в динаміці системи, далекій від рівноваги, включаючи негауссові випадкові процеси та дисипативний хаос.

### 9.1.1 Незворотність на основі діаграм Пуанкаре

Діаграма Пуанкаре для часового ряду являє собою графік, на осі якого розташовані значення для поточного часу , а на осі — його наступні значення в часі . Усі наступні значення, які рівні один одному (), розташовані на лінії ідентичності (line of identity, LI). Інтервали, що представляють зростаючу тендецію, відмічені вище LI (), тоді як спадна тенденція характеризуватиметься скупченням точок нижче LI (). Оцінюючи асиметрію точок на діаграмі, ми можемо вивести різні кількісні показники незворотності (асиметрії) досліджуваних систем.

**Індекс Гузіка (GIx)**

GIx можна визначити як відношення відстаней точок вище LI до відстаней усіх точок на діаграмі:

де позначає кількість точок над LI; позначає кількість точок на графіку Пуанкаре; це відстань від точки над LI до самої LI. Відстань точки до LI можна визначити як

**Індекс Порти**

Індекс Порти (PIx) визначається як кількість точок нижче LI, поділена на загальну кількість точок на графіку Пуанкаре, за винятком тих, що знаходяться на LI:

де кількість точок нижче LI.

**Індекс Кошти**

Індекс Кошти бере до уваги кількість інкриментів () та декриментів (). Вони представляються симетричними, якщо рівні один одному. Даний індекс розраховується для двовимірної мультимасштабної площини (), де новий крос-гранульований ряд для відображає асиметрію інкриментів та декриментів ряду, і індекс незворотності для діапазону масштабів визначається наступним виразом:

Узагальнений індекс Кошти для діапазону мастабів може бути визначений як

де — це максимальний масштаб.

**Індекс Ейлера**

Опираючись на асиметрію розподілу точок нижче та вище LI, Ейлер запропонував індекс асиметрії:

Значне відхилення від 0 вказує на асиметрію системи. Якщо , розподіл точок на діаграмі Пуанкаре значно зміщений у сторону вище LI. Зворотня ситуація спостерігається для . Для досліджувані сегменти представляються зворотніми в часі.

**Індекс площі**

Індекс площі (AIx) визначається як сукупна площа секторів, що сформовані точками над LI поділена на сукупну площу секторів, що відповідають усім точкам на графіку Пуанкаре (крім тих, що розташовані точно на LI). Площа сектора, що відповідає певній точці на графіку Пуанкаре, обчислюється як

де — це радіус сектора; ; — це фазовий кут, і , що визначає фазовий кут -ої точки. Далі, визначається за наступною формулою:

**Індекс кута нахилу**

На додачу до представлених вище мір, було запропоновано розраховувати незворотність сигналу з відношення кутів нахилу точок над LI до нахилу всіх точок на діаграмі:

### 9.1.2 Методи складних мереж

Графи видимості (VG) базуються на простому відображенні часових рядів у мережеву область, використовуючи локальну опуклість скалярно-позначених часових рядів, де кожне спостереження є вершиною в складній мережі. Дві вершини і пов’язані ребром, якщо для всіх вершин застосовується наступна умова:

Матрицю суміжності () представленого ненаправленого та незваженого VG можна представити як:

де — це функція Гевісайда.

Граф горизонтальної видимості (HVG) є спрощеною версією цього алгоритму. Для досліджуваного часового ряду набори вершин VG і HVG однакові, тоді як набір ребер HVG відображає взаємну горизонтальну видимість двох спостережень та . Тобто можна побудувати ребро , якщо для всіх при так що

VG і HVG фіксують по суті одні й ті ж властивості досліджуваної системи, оскільки HVG є підграфом VG з тим же набором вершин, але володіє тільки підмножиною ребер VG. Зверніть увагу, що VG інваріантний щодо суперпозиції лінійних трендів, тоді як HVG — ні.

Оскільки визначення VGs та HVGs чітко враховує часовий порядок спостережень, напрямок часу нерозривно пов’язаний з отриманою структурою мережі. Щоб врахувати цей факт, ми визначаємо набір нових статистичних мережевих показників на основі двох простих характеристик вершин:

* Оскільки кількість ребер інцидентних вершині можна визначити як , для (H)VG ми можемо переписати дану кількісну характеристику для вершини в час відносно її минулих та майбутніх вершин:

де , і та сприймаються як вхідні (минулі) та вихідні (майбутні) вершини.

* Локальний коефіцієнт кластеризації інша властивість старшного порядку структурного сусідства вершини . Для дослідження незворотності, ми можемо переписати дані характеристики наступним чином:

Якщо уявити нашу систему зворотною в часі, ми припускаємо, що розподілу ймовірностей прямих і зворотних за часом характеристик повинні бути однаковими. Для незворотних процесів ми очікуємо виявити статистичну нееквівалентність. Ця нееквівалентність буде визначатися через дивергенцію Кульбака-Лейблера:

де відповідатиме розподілу вхідних характеристикам, а відповідатиме зворотнім.

### 9.1.3 Незворотність на основі пермутаційних шаблонів

Ідея аналізу пермутаційних шаблонів (PP — permutation patterns) спочатку була запропонована Бандтом і Помпе, щоб надати дослідникам простий та ефективний інструмент для характеристики складності динаміки реальних систем. Він уникає порогу амплітуди і замість цього має справу з порядковими шаблонами перестановок. Їх частоти дозволяють відрізнити детерміновані процеси від абсолютно випадкових. Розрахунки PP припускають, що часовий ряд розбивається на пересічні підвектори довжини :

де часова затримка відповідає часу розділення між елементами.

Після цього кожен вектор представляється у вигляді порядкового шаблону , що має задовільняти наступній умові:

Цікава для нас міра незворотності часу на основі PP може бути отримана шляхом врахування їх відносної частоти як для початкового, так і для оберненого часового ряду. Відповідно, якщо обидва типи мають приблизно однакові розподіли ймовірностей своїх патернів, часові ряди представляються зворотними, а для іншого випадку робиться протилежний висновок.

Різницю між розподілами прямих часових рядів () та зворотних () можна оцінити за допомогою дивергенції Кульбака-Лейблера.

## 9.2 Хід роботи

### 9.2.1 Підключення необхідних бібліотек

import numpy as np  
from sklearn import preprocessing  
import matplotlib.pyplot as plt  
import matplotlib.dates as mdates  
import scienceplots  
import pandas as pd  
import yfinance as yf  
import networkx as nx  
import neurokit2 as nk  
  
from scipy.stats import entropy  
from ordpy import ordinal\_distribution  
from tqdm import tqdm  
from scipy.integrate import quad  
from scipy.stats import gaussian\_kde  
from scipy.spatial import distance  
from KDEpy import FFTKDE  
from ts2vg import NaturalVG, HorizontalVG  
  
%matplotlib inline

### 9.2.2 Встановлення параметрів для побудови графіків

plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься  
 # для виведення рисунків  
  
params = {  
 'figure.figsize': (8, 6), # встановлюємо ширину та висоту рисунків за замовчуванням  
 'font.size': 22, # розмір фонтів рисунку  
 'lines.linewidth': 2, # товщина ліній  
 'axes.titlesize': 'small', # розмір титулки над рисунком  
 "font.family": "sans-serif", # сімейство стилів підписів   
 "font.serif": ["Times"], # стиль підпису  
 'savefig.dpi': 300 # якість збережених зображень  
}  
  
plt.rcParams.update(params) # оновлення стилю згідно налаштувань

### 9.2.3 Визначення функції для побудови рекурентного графа

def recurrence\_net(time\_ser, rec\_thr, dim, tau, dist\_type='euclidien'):  
 time\_series = nk.complexity\_embedding(time\_ser, dimension=dim, delay=tau)  
 rp = (distance.cdist(time\_series, time\_series, dist\_type) <= rec\_thr).astype(int)  
 adj\_matrix\_RN = rp  
 np.fill\_diagonal(adj\_matrix\_RN, 0)  
  
 rec\_nw = nx.from\_numpy\_matrix(adj\_matrix\_RN)  
   
 return rec\_nw  
  
def node\_positions\_recurrence\_net(ts, xs):  
 return {i: (xs[i], ts[i]) for i in range(len(ts))}

### 9.2.4 Визначення функції для розрахунку індексу Кошти

def Costa\_1(time\_ser, taus):  
 Cst = []  
 for tau in taus:  
 fragm\_Costa = np.array([time\_ser[tau:], time\_ser[:-tau]])  
 DiffCosta = np.diff(fragm\_Costa,axis=0)  
 IncCosta = np.sum(DiffCosta>0)  
 DecCosta = np.sum(DiffCosta<0)  
 C = (IncCosta-DecCosta)/(len(time\_ser)-tau)  
 Cst.append(C)  
 Costa = np.mean(np.abs(Cst))  
 return Costa

### 9.2.5 Оголошення функцій для підрахунку показників незворотності

#### 9.2.5.1 Пермутаційна незворотність

def PermIrrever(time\_ser, d\_e, tau, delta=1e-10, distance\_irr="kullback"):  
 pattern, dist = ordinal\_distribution(time\_ser, dx=d\_e, taux=tau, return\_missing=True)  
   
 m, n = pattern.shape  
 pf = []  
 pb = []  
  
 is\_used = np.zeros((m))  
  
 for i in range(m):  
 if is\_used[i] == 1:  
 continue  
  
 is\_used[i] = 1  
 pf.append(i)  
 permb = pattern[i,::-1]  
 for j in range(m-1,-1,-1):  
 if np.sum(pattern[j,:] == permb) == n:   
 is\_used[j] = 1  
 pb.append(j)  
 break  
   
 if distance\_irr == "kullback":  
 KLD\_perm = dist[pf] \* np.log((dist[pf] + delta) / (dist[pb] + delta))  
 return np.sum(KLD\_perm)   
 else:  
 return distance.jensenshannon(dist[pf] + delta, dist[pb] + delta)

#### 9.2.5.2 Графо-динамічна незворотність

def GraphIrrever(fragm\_1, graph\_type='classic', delta=1e-10, d\_e\_rec=3, tau\_rec=1, eps\_rec=0.1, dist\_rec='chebyshev', distance\_irr='kullback'):  
   
 # будуємо граф   
   
 if graph\_type == 'classic':  
 g = NaturalVG(directed=None).build(fragm\_1)  
 elif graph\_type == 'horizontal':  
 g = HorizontalVG(directed=None).build(fragm\_1)  
 else:  
 g = recurrence\_net(fragm\_1, rec\_thr=eps\_rec\*np.abs(np.std(fragm\_1)), dim=d\_e\_rec, tau=tau\_rec, dist\_type=dist\_rec)  
   
 # розраховуємо вхідні та вихідні характеристики   
   
 adjacency\_mat = g.adjacency\_matrix()  
 ret\_deg, adv\_deg = GetDegree(adjacency\_mat)  
 ret\_clust, adv\_clust = GetLocalClusteringCoefficient(adjacency\_mat, ret\_deg, adv\_deg)  
   
 # використовуємо kde для знаходження функції щільності ймовірностей   
   
 pdf\_ret\_deg = gaussian\_kde(sorted(ret\_deg), bw\_method='scott')   
 pdf\_adv\_deg = gaussian\_kde(sorted(adv\_deg), bw\_method='scott')   
 pdf\_ret\_clust = gaussian\_kde(sorted(ret\_clust), bw\_method='scott')   
 pdf\_adv\_clust = gaussian\_kde(sorted(adv\_clust), bw\_method='scott')  
   
 a\_deg = min(min(ret\_deg), min(adv\_deg))   
 b\_deg = max(max(ret\_deg), max(adv\_deg))   
 a\_clust = min(min(ret\_clust), min(adv\_clust))  
 b\_clust = max(max(ret\_clust), max(adv\_clust))  
   
 if distance\_irr == 'kullback':  
 dkl\_deg = lambda x: pdf\_ret\_deg.pdf(x) \* np.log((pdf\_ret\_deg.pdf(x) + delta)/(pdf\_adv\_deg.pdf(x) + delta))   
 dkl\_clust = lambda x: pdf\_ret\_clust.pdf(x) \* np.log((pdf\_ret\_clust.pdf(x) + delta)/(pdf\_adv\_clust.pdf(x) + delta))  
  
 distance\_deg = quad(dkl\_deg, a\_deg, b\_deg)[0]   
 distance\_clust = quad(dkl\_clust, a\_clust, b\_clust)[0]  
   
 if distance\_irr == 'shannon':   
 width\_deg = (b\_deg-a\_deg)/len(ret\_deg)  
 width\_clust = (b\_clust-a\_clust)/len(ret\_clust)  
  
 lin\_deg = np.arange(a\_deg, b\_deg, width\_deg)   
 lin\_clust = np.arange(a\_clust, b\_clust, width\_clust)  
  
 p\_ret\_deg = pdf\_ret\_deg.pdf(lin\_deg)   
 p\_adv\_deg = pdf\_adv\_deg.pdf(lin\_deg)   
 p\_ret\_clust = pdf\_ret\_clust.pdf(lin\_clust)   
 p\_adv\_clust = pdf\_adv\_clust.pdf(lin\_clust)  
   
 distance\_deg = distance.jensenshannon(p\_ret\_deg + delta, p\_adv\_deg + delta)  
 distance\_clust = distance.jensenshannon(p\_ret\_clust + delta, p\_adv\_clust + delta)  
   
 return distance\_deg, distance\_clust

#### 9.2.5.3 Функції для отримання ступеня вершини та локальної кластеризації

Процедура знаходження ступеню зв’язків та локальної кластеризації кожної вершини є доволі громіздкою. Для прискорення розрахунків відповідних процедур скористаємось бібліотекою numba. [Numba](https://numba.readthedocs.io/en/stable/index.html) — це швидкий компілятор для Python, який найкраще працює з кодом, що використовує масиви, функції та цикли NumPy. Найпоширеніший спосіб використання Numba — це колекція декораторів, які можна застосувати до ваших функцій, щоб доручити Numba їх компілювати. Коли здійснюється виклик функції, прикрашеної Numba, вона компілюється у машинний код “just-in-time” для виконання, і весь або частина вашого коду може згодом виконуватися зі швидкістю власного машинного коду!

Встановити її можна в наступний спосіб:

!pip install numba==0.56.4

Numba надає декілька утиліт для генерації коду, але центральною функцією є декоратор numba.jit(). За допомогою цього декоратора ви можете позначити функцію для оптимізації JIT-компілятором Numba. Різні режими виклику викликають різні варіанти компіляції та поведінки. Імпортуємо відповідний декоратор з бібліотеки numba:

from numba import jit

@jit(nopython=True, nogil=True)   
def GetDegree(AM):  
 numNodes = AM.shape[0]  
 retarded\_degree = np.zeros((numNodes))  
 advanced\_degree = np.zeros((numNodes))  
   
 for i in range(numNodes):  
 retarded\_degree[i] = AM[i, :i].sum()  
  
 for i in range(numNodes):  
 advanced\_degree[i] = AM[i, i:].sum()  
   
 return retarded\_degree, advanced\_degree  
  
  
@jit(nopython=True, nogil=True)   
def GetLocalClusteringCoefficient(AM, ret\_deg, adv\_deg):  
   
 numNodes = AM.shape[0]  
 retardedCC = np.zeros( (numNodes) )  
 advancedCC = np.zeros( (numNodes) )  
 ret\_norm = ret\_deg \* (ret\_deg - 1) / 2  
 adv\_norm = adv\_deg \* (adv\_deg - 1) / 2  
   
 for i in range(numNodes):  
 if ret\_norm[i] != 0:   
 counter = 0  
   
 for j in range(i):  
 for k in range(j):   
 if AM[i, j] == 1 and AM[j, k] == 1 and AM[k, i] == 1:   
 counter += 1  
   
 retardedCC[i] = counter / ret\_norm[i]  
   
 for i in range(numNodes-2):  
 if adv\_norm[i] != 0:   
 counter = 0  
   
 for j in range(i+1, numNodes):  
 for k in range(i+1, j):   
 if AM[i, j] == 1 and AM[j, k] == 1 and AM[k, i] == 1:   
 counter += 1  
   
 advancedCC[i] = counter / adv\_norm[i]  
   
   
 return retardedCC, advancedCC

### 9.2.6 Завантажуємо дані з сайту Yahoo! Finance

symbol = '^RUT' # Символ індексу  
  
data = yf.download(symbol) # вивантажуємо дані  
time\_ser = data['Adj Close'].copy() # зберігаємо саме ціни закриття  
  
xlabel = 'time, days' # підпис по вісі Ох   
ylabel = symbol # підпис по вісі Оу  
  
date\_in\_num = mdates.date2num(time\_ser.index)  
  
np.savetxt(f'{symbol}\_initial\_time\_series.txt', time\_ser.values)

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

### 9.2.7 Виводимо досліджувані ряди

fig, ax = plt.subplots() # Створюємо порожній графік  
ax.plot(time\_ser.index, time\_ser.values) # Додаємо дані до графіку  
ax.legend([symbol]) # Додаємо легенду  
ax.set\_xlabel(xlabel) # Встановимо підпис по вісі Ох  
ax.set\_ylabel(ylabel) # Встановимо підпис по вісі Oy  
  
plt.xticks(rotation=45) # оберт позначок по осі Ох на 45 градусів  
  
plt.savefig(f'{symbol}.jpg') # Зберігаємо графік   
plt.show(); # Виводимо графік

|  |
| --- |
| Рисунок 9.1: Динаміка щоденних змін індексу Russell 2000 |

Користуючись тими методами, що ми розглянули в попередній лабораторній роботі, побудуємо діаграму Пуанкаре та граф нашого часового ряду. Але перш за все, для діаграми Пункаре, треба знайти стандартизовані прибутковості. Для цього оголосимо функцію transformation(), що прийматиме на вхід часовий сигнал, тип ряду, і повертатиме його перетворення:

def transformation(signal, ret\_type):  
  
 for\_rec = signal.copy()  
  
 if ret\_type == 1: # Зважаючи на вид ряду, виконуємо  
 # необхідні перетворення  
 pass  
 elif ret\_type == 2:  
 for\_rec = for\_rec.diff()  
 elif ret\_type == 3:  
 for\_rec = for\_rec.pct\_change()  
 elif ret\_type == 4:  
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 elif ret\_type == 5:   
 for\_rec = for\_rec.pct\_change()  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
 for\_rec = for\_rec.abs()  
 elif ret\_type == 6:  
 for\_rec -= for\_rec.mean()  
 for\_rec /= for\_rec.std()  
  
 for\_rec = for\_rec.dropna().values  
  
 return for\_rec

### 9.2.8 Встановлення параметрів для розрахунків

window = 500 # розмір ковзного вікна  
tstep = 1 # часовий крок  
  
ret\_type = 4 # тип ряду: 1 - вихідний, 2 - детрендований  
 # 3 - стандартні прибутковості,   
 # 4 - стандартизовані прибутковості,   
 # 5 - абсолютні значення (волатильності)  
 # 6 - стандартизований вихідний часовий ряд  
  
  
# параметри для рекурентного графу  
d\_e\_rec = 3 # розмірність вкладень  
tau\_rec = 1 # часова затримка  
eps\_rec = 1.3 # радіус  
dist\_rec = 'chebyshev' # відстань між траєкторіями:   
 # canberra’, ‘chebyshev’, ‘cityblock’, ‘correlation’,   
 # ‘cosine’, ‘dice’, ‘euclidean’, ‘hamming’, ‘jaccard’,   
 # ‘jensenshannon’, ‘kulsinski’, ‘kulczynski1’, ‘mahalanobis’,   
 # ‘matching’, ‘minkowski’, ‘rogerstanimoto’, ‘russellrao’, ‘seuclidean’,   
 # ‘sokalmichener’, ‘sokalsneath’, ‘sqeuclidean’, ‘yule’.  
  
  
# параметри для мір незворотності  
d\_e\_perm = 3 # розмірність вкладень для пермутаційних патернів   
tau\_perm = 1 # часова затримка для пермутаційних патернів  
distance\_irr = 'kullback' # відстань між розподілами: kullback, shannon  
graph\_type = 'classic' # тип графу: classic, horizontal, recurrent  
  
# параметри для мір асиметрії  
tau\_assym = 1 # часова затримка для діаграми Пуанкаре  
tau\_Costa\_begin = 1 # початковий часовий масштаб для індексу Кошти  
tau\_Costa\_end = 20 # кінцевий часовий масштаб для індексу Кошти  
taus\_Costa = np.arange(tau\_Costa\_begin, tau\_Costa\_end+1) # формуємо масив масштабів  
   
length = len(time\_ser)

### 9.2.9 Виводимо діаграму Пуанкаре та розраховуємо міри на її основі

for\_puank = time\_ser.copy()  
  
for\_puank = transformation(for\_puank, ret\_type)

fig, ax1 = plt.subplots(1, 1)  
  
ax1.scatter(for\_puank[:-tau\_assym],for\_puank[tau\_assym:], marker="X", s=180, c="g")  
  
low\_x, high\_x = ax1.get\_xlim()  
low\_y, high\_y = ax1.get\_ylim()  
ax1.axline([low\_x, low\_y], [high\_x, high\_y])  
  
ax1.set\_aspect('equal', 'box')  
ax1.set\_xlabel(r'$g(t)$')  
ax1.set\_ylabel(r'$g(t+\tau)$')   
ax1.set\_xlim(left=low\_x, right=high\_x)  
ax1.set\_ylim(bottom=low\_y, top=high\_y)  
plt.locator\_params(axis='y', nbins=7)  
  
plt.savefig(f"Poincare\_plot\_{symbol}\_{tau\_assym}\_{window}\_{tstep}.jpg", bbox\_inches="tight")  
plt.show()



Виходячи з даної діаграми, можна зазначити, що для прибутковостей індексу Russell 2000 спостерігається асиметрія у сторону зростаючих флуктуацій ряду.

### 9.2.10 Побудова показників незворотності із використанням ковзного вікна

Визначаємо функцію для побудови парних графіків:

def plot\_pair(x\_values,   
 y1\_values,  
 y2\_values,   
 y1\_label,   
 y2\_label,  
 x\_label,   
 file\_name, clr="magenta"):  
  
 fig, ax = plt.subplots()  
  
 ax2 = ax.twinx()  
  
 ax2.spines.right.set\_position(("axes", 1.03))  
  
 p1, = ax.plot(x\_values,   
 y1\_values,   
 "b-", label=fr"{y1\_label}")  
 p2, = ax2.plot(x\_values,  
 y2\_values,   
 color=clr,   
 label=y2\_label)  
  
 ax.set\_xlabel(x\_label)  
 ax.set\_ylabel(f"{y1\_label}")  
  
 ax.yaxis.label.set\_color(p1.get\_color())  
 ax2.yaxis.label.set\_color(p2.get\_color())  
  
 tkw = dict(size=2, width=1.5)  
  
 ax.tick\_params(axis='x', \*\*tkw)  
 ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
 ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
  
  
 ax2.legend(handles=[p1, p2])  
  
 plt.savefig(file\_name + ".jpg")  
   
 plt.show();

Оголошуємо масиви для збереження результатів:

PIx = []  
GIx = []  
SIx = []  
AIx = []  
EIx = []  
CIx = []

Розраховуємо відповідні міри у віконній процедурі:

for i in tqdm(range(0,length-window,tstep)):  
 fragm = time\_ser.iloc[i:i+window].copy() # відбираємо фрагмент ряду   
   
 fragm = transformation(fragm, ret\_type)  
   
 Temp\_fragm = np.array([fragm[:-tau\_assym], fragm[tau\_assym:]])  
   
 T2 = np.transpose(np.arctan(Temp\_fragm[1,:]/Temp\_fragm[0,:])\*180/np.pi)  
 Dup = abs(np.diff(Temp\_fragm[:,T2>45],axis=0))  
 Dtot = abs(np.diff(Temp\_fragm[:,T2!=45],axis=0))  
 Sup = np.sum(abs(T2[T2>45]-45))  
 Stot = np.sum(abs(T2[T2!=45]-45))  
 Aup = np.sum(abs(np.transpose(((T2[T2>45]-45))\*np.sqrt(np.sum(Temp\_fragm[:,T2>45]\*\*2,axis=0)))))  
 Atot = np.sum(abs(np.transpose(((T2[T2!=45]-45))\*np.sqrt(np.sum(Temp\_fragm[:,T2!=45]\*\*2,axis=0)))))  
 Ethird = np.sum(np.transpose(Temp\_fragm[0,:]-Temp\_fragm[1,:])\*\*3)  
 Etot = (np.sum(np.transpose(Temp\_fragm[0,:]-Temp\_fragm[1,:])\*\*2))\*\*(3/2)  
  
   
 Porta = sum(T2<45)/sum(T2!=45)  
 Gudzik = np.sum(Dup\*\*2)/np.sum(Dtot\*\*2)  
 Slope = Sup/Stot  
 Area = Aup/Atot  
 Eiler = Ethird/Etot  
 Costa = Costa\_1(fragm, taus\_Costa)  
   
 PIx.append(Porta)  
 GIx.append(Gudzik)  
 SIx.append(Slope)  
 AIx.append(Area)  
 EIx.append(Eiler)  
 CIx.append(Costa)

100%|██████████| 8569/8569 [00:14<00:00, 600.95it/s]

Зберігаємо значення до .txt файлів

np.savetxt(f"Porta\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}\_{tau\_assym}.txt", PIx)  
np.savetxt(f"Gudzik\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}\_{tau\_assym}.txt", GIx)  
np.savetxt(f"Slope\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}\_{tau\_assym}.txt", SIx)  
np.savetxt(f"Area\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}\_{tau\_assym}.txt", AIx)  
np.savetxt(f"Eiler\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}\_{tau\_assym}.txt", EIx)  
np.savetxt(f"Costa\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}\_{tau\_assym}.txt", CIx)

### 9.2.11 Візуалізація показників на основі діаграми Пуанкаре

#### 9.2.11.1 Індекс Порти

measure\_label = r"$PIx$"  
file\_name = f"PIx\_{symbol}\_{tau\_assym}\_{window}\_{tstep}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 PIx,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name,   
 clr="crimson")

|  |
| --- |
| Рисунок 9.2: Динаміка індексу Russell 2000 та індексу Порти |

#### 9.2.11.2 Індекс Гузіка

measure\_label = r"$GIx$"  
file\_name = f"GIx\_{symbol}\_{tau\_assym}\_{window}\_{tstep}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 GIx,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name,   
 clr="crimson")

|  |
| --- |
| Рисунок 9.3: Динаміка індексу Russell 2000 та індексу Гузіка |

#### 9.2.11.3 Індекс кута нахилу

measure\_label = r"$SIx$"  
file\_name = f"SIx\_{symbol}\_{tau\_assym}\_{window}\_{tstep}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 SIx,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name,   
 clr="crimson")

|  |
| --- |
| Рисунок 9.4: Динаміка індексу Russell 2000 та індексу кута нахилу |

#### 9.2.11.4 Індекс площі секторів

measure\_label = r"$AIx$"  
file\_name = f"AIx\_{symbol}\_{tau\_assym}\_{window}\_{tstep}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 AIx,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name,   
 clr="crimson")

|  |
| --- |
| Рисунок 9.5: Динаміка індексу Russell 2000 та індексу площі секторів |

#### 9.2.11.5 Індекс Ейлера

measure\_label = r"$EIx$"  
file\_name = f"EIx\_{symbol}\_{tau\_assym}\_{window}\_{tstep}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 EIx,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name,   
 clr="crimson")

|  |
| --- |
| Рисунок 9.6: Динаміка індексу Russell 2000 та індексу Ейлера |

#### 9.2.11.6 Індекс Кошти

measure\_label = r"$CIx$"  
file\_name = f"CIx\_{symbol}\_{tau\_assym}\_{window}\_{tstep}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 CIx,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name,   
 clr="crimson")

|  |
| --- |
| Рисунок 9.7: Динаміка індексу Russell 2000 та індексу Кошти |

### 9.2.12 Побудова графу досліджуваного ряду

index\_begin = 2000 # початковий індекс для графу  
index\_end = 4000 # кінцевий індекс для графу  
  
ret\_type = 1 # вид ряду  
  
for\_graph\_plot = time\_ser.copy()  
  
for\_graph\_plot = transformation(for\_graph\_plot, ret\_type)  
  
date = date\_in\_num[index\_begin:index\_end] # вилучаємо необхідні по індексам дати  
  
# будуємо граф у залежності від типу графа  
if graph\_type == 'classic':  
 g = NaturalVG(directed=None).build(for\_graph\_plot[index\_begin:index\_end], xs=date)  
 pos = g.node\_positions()  
 nxg = g.as\_networkx()  
elif graph\_type == 'horizontal':  
 g = HorizontalVG(directed=None).build(for\_graph\_plot[index\_begin:index\_end], xs=date)  
 pos = g.node\_positions()  
 nxg = g.as\_networkx()  
else:  
 g = recurrence\_net(for\_graph\_plot[index\_begin:index\_end],   
 rec\_thr=eps\_rec \* np.abs(np.std(for\_graph\_plot[index\_begin:index\_end])),   
 dim=d\_e\_rec,   
 tau=tau\_rec,   
 dist\_type=dist\_rec)  
   
 pos = node\_positions\_recurrence\_net(for\_graph\_plot[index\_begin:index\_end], date)  
 nxg = g  
   
   
# встановлення параметрів для побудови графів  
  
graph\_plot\_options = {  
 'with\_labels': False,  
 'node\_size': 2,  
 'node\_color': [(0, 0, 0, 1)],  
 'edge\_color': [(0, 0, 0, 0.15)],  
}

Виводимо зв’язки видимості:

fig, ax1 = plt.subplots(1, 1)  
  
nx.draw\_networkx(nxg, ax=ax1, pos=pos, \*\*graph\_plot\_options)  
ax1.tick\_params(bottom=True, labelbottom=True)  
ax1.plot(time\_ser.index[index\_begin:index\_end],   
 for\_graph\_plot[index\_begin:index\_end],   
 label=fr"{symbol}")  
  
ax1.set\_title('Visibility Connections', fontsize=22)  
ax1.set\_xlabel(xlabel)  
ax1.set\_ylabel(fr"{symbol}")  
ax1.legend(loc='upper right')  
  
plt.savefig(f"Time\_ser\_connections\_symbol={symbol}\_ \  
 idx\_beg={index\_begin}\_idx\_end={index\_end}\_sertype={ret\_type}\_ \  
 network\_type={graph\_type}.jpg", bbox\_inches="tight")  
  
plt.show();



Виходячи з графу взятого нами фрагменту видно, що крах поблизу 2000-го року характеризується високою концентрацією вузлів. Це вказує на високий ступінь довготривалої пам’яті для кризових явищ фондового ринку, що в свою чергу впливає і на їх незворотність.

Тепер розглянемо сам граф:

pos = nx.spring\_layout(nxg, k=0.15, iterations=100)  
# знаходимо вузол близький до центру графа (0.5,0.5)  
dmin = 1  
ncenter = 0  
for n in pos:   
 x, y = pos[n]  
 d = (x - 0.5)\*\*2 + (y - 0.5)\*\*2  
 if d < dmin:  
 ncenter = n  
 dmin = d  
  
# розфарбовуємо в залежності від ступеня вершини  
  
p = dict(nx.degree(nxg))  
fig, ax2 = plt.subplots(1, 1)  
ax2.set\_title('Graph representation')  
nx.draw\_networkx\_edges(nxg, ax=ax2, pos=pos, nodelist=[ncenter], alpha=0.4,width=0.1)  
nx.draw\_networkx\_nodes(nxg, ax=ax2, pos=pos, nodelist=list(p.keys()),  
 node\_size=10, edgecolors='r', linewidths=0.01,  
 node\_color=list(p.values()),  
 cmap=plt.cm.Blues\_r)  
   
vmin = np.asarray(list(p.values())).min()  
vmax = np.asarray(list(p.values())).max()  
  
sm = plt.cm.ScalarMappable(cmap=plt.cm.Blues\_r, norm=plt.Normalize(vmin=vmin, vmax=vmax))  
cb = plt.colorbar(sm, ax=ax2)  
cb.set\_label('degree')  
  
plt.savefig(f"Graph\_representation\_symbol={symbol}\_ \  
 idx\_beg={index\_begin}\_idx\_end={index\_end} \  
 \_sertype={ret\_type}\_network\_type={graph\_type}.jpg", bbox\_inches="tight")  
plt.show();



### 9.2.13 Побудова показників незворотності на основі пермутаційних шаблонів та графів

Ініціалізуємо масиви для збереження результатів розрахунків:

Degree = []  
Clust = []  
Perm = []

Розпочинаємо процедуру рухомого вікна:

for i in tqdm(range(0,length-window,tstep)):  
 fragm = time\_ser.iloc[i:i+window].copy() # відбираємо фрагмент ряду  
  
 fragm = transformation(fragm, ret\_type) # виконуємо перетворення  
   
 deg, clust = GraphIrrever(fragm,   
 graph\_type=graph\_type,   
 delta=1e-10,   
 d\_e\_rec=d\_e\_rec,   
 tau\_rec=tau\_rec,   
 eps\_rec=eps\_rec,   
 dist\_rec=dist\_rec,   
 distance\_irr=distance\_irr)   
   
 perm = PermIrrever(fragm,   
 d\_e=d\_e\_perm,   
 tau=tau\_perm,   
 delta=1e-10,   
 distance\_irr=distance\_irr)  
   
 Degree.append(deg)  
 Clust.append(clust)  
 Perm.append(perm)

100%|██████████| 8569/8569 [16:54<00:00, 8.45it/s]

Зберігаємо результати до .txt файлів

np.savetxt(f"{distance\_irr}\_deg\_symbol={symbol}\_wind={window} \  
 \_step={tstep}\_ret\_type={ret\_type}\_graph\_type={graph\_type}.txt", Degree)  
np.savetxt(f"{distance\_irr}\_clust\_symbol={symbol}\_wind={window} \  
 \_step={tstep}\_ret\_type={ret\_type}\_graph\_type={graph\_type}.txt", Clust)  
np.savetxt(f"{distance\_irr}\_perm\_symbol={symbol}\_wind={window} \  
 \_step={tstep}\_ret\_type={ret\_type}\_d\_e={d\_e\_perm}\_tau={tau\_perm}.txt", Perm)

### 9.2.14 Візуалізація показників на основі графів та пермутаційних шаблонів

#### 9.2.14.1 Ступінь незворотності на основі ступеня вершини

measure\_label = r"$Dist\_{deg}$"  
file\_name = f"Degree\_symbol={symbol}\_wind={window}\_ \  
 step={tstep}\_ret\_type={ret\_type} \  
 \_graph\_type={graph\_type}\_dist={distance\_irr}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 Degree,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name,   
 clr="darkgreen")

|  |
| --- |
| Рисунок 9.8: Динаміка індексу Russell 2000 та показника незворотності на основі ступеня вершини |

#### 9.2.14.2 Ступінь незворотності на основі показника локальної кластеризації

measure\_label = r"$Dist\_{clust}$"  
file\_name = f"Clust\_symbol={symbol}\_wind={window}\_ \  
 step={tstep}\_ret\_type={ret\_type} \  
 \_graph\_type={graph\_type}\_dist={distance\_irr}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 Clust,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name,   
 clr="darkgreen")

|  |
| --- |
| Рисунок 9.9: Динаміка індексу Russell 2000 та показника незворотності на основі локальної кластеризації |

#### 9.2.14.3 Ступінь незворотності на основі пермутаційних шаблонів

measure\_label = r"$Dist\_{clust}$"  
file\_name = f"Clust\_symbol={symbol}\_wind={window}\_ \  
 step={tstep}\_ret\_type={ret\_type} \  
 d\_e={d\_e\_perm}\_tau={tau\_perm} \  
 \_graph\_type={graph\_type}\_dist={distance\_irr}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 Perm,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name,   
 clr="darkgreen")

|  |
| --- |
| Рисунок 9.10: Динаміка індексу Russell 2000 та показника незворотності на основі пермутаційних шаблонів |

## 9.3 Висновок

У даній роботі було розглянуто показники незворотності (асиметрії) системи на основі діаграм Пуанкаре, графу видимості та пермутаційних шаблонів. Було продемонстровано побудову діграми Пуанкаре та зв’язків видимості як для всього ряду, так і для деяких із його фрагментів. Видно, що значення на діаграмі Пуанкаре характеризуються розподілом точок, що виходять за межі нормального Гаусового розподілу. Для графу видимості видно, що кризові стани характеризуються значною концентрацією зв’язків, що є довгостроковими. Таким чином, поставало актуальним проводити розрахунок показників незворотності графового типу для вихідного ряду.

Показники на основі діаграми Пуанкаре демонструють зріст або спад у кризові періоди, що вказує на зростання асиметрії у даний період часу. Дані показники можуть слугувати в якості індикаторів кризових явищ.

Виходячи з 3 показників незворотності, що представлені вище, видно, що дані показники починають зростати в передкризовий період, вказуючи на стартовий період хаосу. Найгірше серед них себе поводить . Найращим чином та . Їх і варто використовувати в якості індикаторів-передвісників крахів.

# 10. Лабораторна робота № 12

**Тема.** Важкі хвости: вплив одиничних катастрофічних подій. Аналіз кризових подій із використанням Леві альфа-стабільного розподілу.

**Мета.** Навчитися використовувати розподіл Леві для диференціації флуктуацій, що виходять за межі нормального розподілу.

|  |
| --- |
| Анотація |
| У цій лабораторній ми досліджуємо розподіл прибутковості фінансових активів і стабільність певних показників, які ми розраховуємо, опираючись на ці активи. Зокрема, ми побачимо, що більш високі моменти розподілу прибутковостей з важкими хвостами насправді дуже нестабільні, нескінченні або взагалі не визначені, і що, наприклад, одна єдина точка даних визначає більшу частину надлишкового ексцесу. Цей факт має досить драматичні наслідки, оскільки популярні моделі, такі як GARCH, базуються на можливості оцінки кінцевого значення ексцесу. Суть, яку ми хочемо донести, полягає в тому, що, не приділяючи особливої уваги повноті фінансових розподілів, не можна очікувати, що моделі будуть працювати за межами навчальної та тестувальної вибірки. Аналітики та трейдери завжди будуть дивуватись новим екстремальним подіям, що, як правило, рідко бувають позитивними. |

## 10.1 Теоретичні відомості

### 10.1.1 Імпортуємо необхідні бібліотеки

import numpy as np # бібліотека для роботи з масивами чисел  
import matplotlib.pyplot as plt # бібліотека для побудови графіків  
import yfinance as yf # бібліотека для зчитування фінансових даних з Yahoo Finance  
import levy # бібліотека для роботи з альфа-стабільним розподілом Леві  
import pandas as pd # бібліотека для фільтрації даних та їх обробки  
import scienceplots  
from scipy.stats import norm, laplace # бібліотека для побудови теоретичного розподілу Гауса  
 # та Лапласа  
from tqdm import tqdm # бібліотека для виводу шкали завантаження  
  
%matplotlib inline

### 10.1.2 Виконуємо налаштування рисунків

plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься  
 # для виведення рисунків  
  
params = {  
 'figure.figsize': (8, 6), # встановлюємо ширину та висоту рисунків за замовчуванням  
 'font.size': 14, # розмір фонтів рисунку  
 'lines.linewidth': 2, # товщина ліній  
 'axes.titlesize': 'small', # розмір титулки над рисунком  
 "font.family": "sans-serif", # сімейство стилів підписів   
 "font.serif": ["Times"], # стиль підпису  
 'savefig.dpi': 300 # якість збережених зображень  
}  
  
plt.rcParams.update(params) # оновлення стилю згідно налаштувань

### 10.1.3 Зчитування даних

df = pd.read\_csv('databases\^spx\_d.csv')  
df.set\_index('Date', inplace=True)  
df.index = pd.to\_datetime(df.index)  
df = df[df.index >= '1950-01-01']

### 10.1.4 Важкі хвости та Чорні понеділки

Спочатку ми розглянемо розподіл щоденних прибутковостей індексу S&P 500, починаючи з 1950 року. У клітинці нижче буде показано [Pandas dataframe](https://pandas.pydata.org/docs/user_guide/10min.html) df, що містить дані OHLC з файлу (ці дані були попередньо завантажені в клітинці вище).

df

|  | Open | High | Low | Close | Volume |
| --- | --- | --- | --- | --- | --- |
| Date |  |  |  |  |  |
| 1950-01-03 | 16.66 | 16.66 | 16.66 | 16.66 | 7.000000e+05 |
| 1950-01-04 | 16.85 | 16.85 | 16.85 | 16.85 | 1.050000e+06 |
| 1950-01-05 | 16.93 | 16.93 | 16.93 | 16.93 | 1.416667e+06 |
| 1950-01-06 | 16.98 | 16.98 | 16.98 | 16.98 | 1.116667e+06 |
| 1950-01-07 | 17.09 | 17.09 | 17.09 | 17.09 | 1.116667e+06 |
| ... | ... | ... | ... | ... | ... |
| 2023-04-10 | 4085.20 | 4109.50 | 4072.55 | 4109.11 | 1.951642e+09 |
| 2023-04-11 | 4110.29 | 4124.26 | 4102.61 | 4108.94 | 2.000949e+09 |
| 2023-04-12 | 4121.72 | 4134.37 | 4086.94 | 4091.95 | 2.249009e+09 |
| 2023-04-13 | 4100.04 | 4150.26 | 4099.40 | 4146.22 | 2.198579e+09 |
| 2023-04-14 | 4140.11 | 4163.19 | 4113.20 | 4137.64 | 2.088609e+09 |

Цей історичний ряд щоденних цін містить ~ 18527 торгових днів. Замість того, щоб дивитися безпосередньо на ціни, ми розглянемо щоденні [*log*-прибутковості](https://lucaslouca.com/Why-Use-Logarithmic-Returns-In-Time-Series-Modelling/) індексу S&P 500. Пам’ятайте, що логарифмічна прибутковість визначається як логарифм відношення між цінами закриття у теперішній момент часу та попередній, :

Логарифмічні прибутковості характеризуються симетрією, тобто якщо в один день індекс втрачає , а на наступний день індекс набирає , індекс знову досягає початкової ціни. Зі звичайними прибутковостями, спочатку втративши , а потім знову набравши , ви не повернетесь до початкової ціни. Ми зберігаємо прибутковості в новому фреймі даних lr:

lr = np.log(df['Close']).diff().dropna()

lr.plot()  
plt.show();



Використовуючи вбудовані функції Pandas та Numpy, тепер ми можемо подивитись на найгірше щоденнє значення прибутковостей. Ми виявляємо, що найгірше значення становило , що відповідає , і що це сталося в понеділок, 19 жовтня 1987 року — так званий [*Чорний понеділок*](https://en.wikipedia.org/wiki/Black_Monday_(1987)).

# найгірший щоденні прибутковості від закриття до закриття  
lr.min()

-0.2289972265656708

# відповідні арифметичні прибутковості  
np.exp(lr.min()) - 1

-0.20466926070038938

# індекс найгіршого значення  
np.argmin(lr)

9583

df.iloc[9583:9585]

|  | Open | High | Low | Close | Volume |
| --- | --- | --- | --- | --- | --- |
| Date |  |  |  |  |  |
| 1987-10-16 | 298.08 | 298.92 | 281.52 | 282.70 | 188055556.0 |
| 1987-10-19 | 282.70 | 282.70 | 224.83 | 224.84 | 335722222.0 |

Далі ми будуємо графік вибіркового розподілу щоденних логарифмічних прибутковостей, що демонструє нам, як часто ми спостерігаємо щоденні прибутковості певного знака та величини. Зверніть увагу, що ми встановили логарифмічне масштабування для вісі Оy, щоб мати краще представлення про хвости розподілу, тобто дуже великі, але рідкісні негативні та позитивні прибутковості — крахи та підйоми:

plt.figure()  
plt.hist(lr, 100, density=True)  
plt.yscale('log')  
plt.ylabel('Розп. ймовірностей')  
plt.xlabel('Щоденні лог-прибутковості')  
plt.xlim([-0.25, 0.125])  
plt.ylim([0.011, 99])  
plt.annotate('Чорний понеділок (1987)', [-0.226, 0.02], xytext=[-0.215, 0.15],  
 arrowprops=dict(facecolor='0.4', width=1, headwidth=6, headlength=6),  
 ha='left', va='bottom');  
plt.show();



Як ми можемо описати погіршення розподілу прибутковості? За допомогою [логарифмічного масштабування](https://en.wikipedia.org/wiki/Logarithmic_scale) осі Oy, щоб експоненціальний спад виглядав лінійно, тому ми могли би запідозрити, що при малих абсолютних значеннях прибутковості розподіл зменшується експоненціально, але в міру збільшення абсолютних значень прибутковостей ми все більше і більше відхиляємося від початкового нахилу. Насправді, ми можемо показати, що ні розповсюдженний [розподіл Гауса](https://en.wikipedia.org/wiki/Normal_distribution) (також званий *нормальним* розподілом), ні [розподіл Лапласа](https://en.wikipedia.org/wiki/Laplace_distribution) з його експоненціальним спадом з обох сторін точно не можуть змоделювати розподіл прибутковостей S&P 500. Як розподіл Гауса, так і розподіл Лапласа недооцінюють ймовірність екстремальних подій — підйомів та крахів:

# підганяємо розподіл Гауса та Лапласа під дані  
norm\_loc, norm\_scale = norm.fit(lr)  
laplace\_loc, laplace\_scale = laplace.fit(lr)  
  
# генеруємо значення (x,y) з використанням отриманих параметрів розподілів  
x\_fit = np.linspace(-0.25, 0.125, 1000)  
y\_fit\_norm = norm(loc=norm\_loc, scale=norm\_scale).pdf(x\_fit)  
y\_fit\_laplace = laplace(loc=laplace\_loc, scale=laplace\_scale).pdf(x\_fit)

plt.figure()  
plt.hist(lr, 100, density=True)  
plt.plot(x\_fit, y\_fit\_laplace, lw=3, c='C3', alpha=0.8, label='Розподіл Лапласа')  
plt.plot(x\_fit, y\_fit\_norm, lw=3, c='0.2', alpha=0.8, label='Розподіл Гауса')  
plt.yscale('log')  
plt.ylabel('Розп. ймовірностей')  
plt.xlabel('Щоденні лог-прибутковості')  
plt.xlim([-0.25, 0.125])  
plt.ylim([0.011, 99])  
plt.legend()  
  
plt.annotate('Чорний понеділок (1987)', [-0.226, 0.02], xytext=[-0.215, 0.15],  
 arrowprops=dict(facecolor='0.4', width=1, headwidth=6, headlength=6),  
 ha='left', va='bottom');  
  
plt.show();



### 10.1.5 Відхилення в 23 сигма

Як ми можемо бачити вище, як розподіл Гауса, так і розподіл Лапласа різко недооцінюють ймовірність дуже волатильної динаміки! Часто зазначають, що трапилась “[3-сигма](https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule) подія на ринку” або щось подібне, тому що багатьом людям подобається оцінювати індивідуальні прибутковості по [стандартному відхиленню](https://en.wikipedia.org/wiki/Standard_deviation) (зване “сигмою”) підібраного гаусового розподілу. Однак, якщо Розподіл Гауса взагалі не відповідає розподілу вибірки, як у нашому випадку, оцінка краху в одиницях “сигма” може ввести в оману! Щоб показати це, ми спочатку перевіримо, на скільки сигм ринок змістився в Чорний понеділок 1987 року:

np.abs(np.min(lr))/np.std(lr, ddof=1)

22.969393251617504

Виходячи з розподілу по Гауса, Чорний понеділок був би подією з “23 сигмами”! Тепер Розподіл Гауса чітко говорить нам, як часто X-сигма подія має відбуватися:

* 1-сигма: прибл. 1 з 3 днів
* 2-сигма: прибл. 1 з 22 днів
* 3-сигма: прибл. 1 з 370 днів

…

Ми можемо розширити цю таблицю, попрацювавши з [функцією кумулятивного розподілу](https://en.wikipedia.org/wiki/Cumulative_distribution_function), тісно пов’язана з [функцією виживання](https://en.wikipedia.org/wiki/Survival_function), і перемістивши обчислення в логарифмічний простір для чисельної стабільності, як демонструють наступні комірки коду:

# у середньому, як багато днів між 1-сигма подіями?  
1/(2\*(1. - norm.cdf(1.)))

3.1514871875343764

# у середньому, як багато днів між 2-сигма подіями?  
1/(2\*(1. - norm.cdf(2.)))

21.97789450799283

# у середньому, як багато днів між 3-сигма подіями?  
1/(2\*(1. - norm.cdf(3.)))

370.3983473449564

# те саме, що й вище, але із використанням функції виживання  
1/(2\*norm.sf(3.))

370.3983473449592

# те саме, що й вище, але із використанням логарифмічної функції функції виживання для чисельної стабільності  
np.exp(-norm.logsf(3.) - np.log(2.))

370.398347344959

# те саме, що й вище, але із використанням логарифму з основою 10 для замість натурального логарифму для кращої інтерпретації  
10\*\*(-norm.logsf(3.)/np.log(10) - np.log10(2.))

370.3983473449588

# як багато днів (log10 значення) між 23-сигма подіями, в середньому?  
(-norm.logsf(23.)/np.log(10) - np.log10(2.))

116.33149536636726

Щоб отримати додаткову інформацію про частоту екстремальних прибутковостей, ми можемо навіть створити невеликий [інтерактивний віджет](https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20Basics.html) за допомогою повзунка, який вказує кількість “сигм”, а текст поруч із ним показує середню кількість днів між двома такими екстремальними подіями:

from ipywidgets import IntSlider, interact  
  
def years\_between(X):  
 log10\_days = (-norm.logsf(X)/np.log(10) - np.log10(2.))  
 days = int(10\*\*log10\_days)  
 return f'У сереньому, між двома X-сигма подіями, варто очікувати {days} дні(в), (день).'  
  
interact(years\_between, X=IntSlider(min=1, max=23, step=1, value=1));

interactive(children=(IntSlider(value=1, description='X', max=23, min=1), Output()), \_dom\_classes=('widget-int…

np.log10(float(214533622638557983431869220329015643677794364342592664885632499326649056583365324378194575664549945880382591666749440))

116.33149536636726

Ви бачите, що при переміщенні повзунка вправо кількість днів або років між двома наступними “X-сигма” подіями швидко наростає. Наша таблиця тепер виглядає наступним чином:

* 1-сигма: прибл. 1 з 3 днів
* 2-сигма: прибл. 1 з 22 днів
* 3-сигма: прибл. 1 з 370 днів

…

* 23-сигма: приблизно 1 з днів (!!!)

Внесемо ясність: виходячи з розподілу Гауса, щоденні втрати Чорної п’ятниці 1987 року повинні бути подією, яку ми очікуємо раз на днів. Це число в значній мірі незбагненно велике. Якби S&P 500 почав торгуватися відразу після [народження Всесвіту](https://en.wikipedia.org/wiki/Age_of_the_universe), це становило б лише торгових днів. Через днів усі зірки у Всесвіті давно згорять, навіть усі [чорні діри випаруються](https://en.wikipedia.org/wiki/Future_of_an_expanding_universe#Dark_Era_and_Photon_Age) і Всесвіт стане темним і порожнім місцем. Тож або ми повинні бути дуже щасливі, що єдина очікувана Чорна п’ятниця в історії та майбутньому Всесвіту позаду, або ви дійсно ніколи не повинні використовувати Розподіл Гауса для моделювання прибутковості акцій! Незалежно від ринку та деталей, якщо хтось говорить про події з 10 сигмами або 23 сигмами, він, безумовно, використовує неправильну модель, оскільки шанс спостерігати таку подію в нашому житті незначний.

Отже, виникає питання: Як ми справляємось із такими екстремальними значеннями? Чи слід позначати їх як [викиди](https://en.wikipedia.org/wiki/Outlier) або артефакти, щоб наші фінансові моделі краще описували більшість значень? Звичайно, ні, оскільки результат наших інвестицій може критично залежати не від більшості менших прибутковостей, а саме від таких екстремальних подій! Далі ми познайомимося з підходом, який може пояснити та екстраполювати за межі екстремальних подій, таких як Чорний понеділок. Але ми також побачимо, що не всі способи врахування важких хвостів розподілів працюють добре, оскільки деякі моделі, такі як [GARCH](https://en.wikipedia.org/wiki/Autoregressive_conditional_heteroskedasticity) може заколисати нас помилковим почуттям безпеки, коли ми завжди беремо до уваги минулі екстремальні події, але завжди дивуємося новим.

### 10.1.6 Статистика степеневого розподілу

Тепер, коли ми знаємо, що розподіл Гауса не є хорошим вибором, який тип розподілу насправді може описати частоту екстремальних подій, які струшують S&P 500? Частота екстремально позитивні та негативні прибутковості асиметричні для більшості фінансових активів, оскільки крахи, як правило, більш сильні, ніж підйоми. Ось чому нижче ми зосередимося на лівому (негативному) хвості розподілу прибутковостей.

Оскільки нас цікавлять лише екстремальні події, ми розглянемо лише ті прибутковості, які менше (приблизно три стандартних відхилення від середнього значення). Нижче ми візуалізуємо ці екстремально негативні показники прибутковостей в [логарифмічному масштабі](https://en.wikipedia.org/wiki/Log%E2%80%93log_plot) (тут ми використовуємо абсолютні значення негативних прибутковостей). Як вісь Oy (яка показує частоту логарифмічних прибутковостей), так і вісь Ox (яка показує величину логарифмічних прибутковостей) масштабуються логарифмічно:

# хвіст починається приблизно при трьох стандартних відхиленнях від середнього  
np.mean(lr) - 3\*np.std(lr)

-0.029610514264246313

neg\_lr = -lr[lr < 0]  
  
plt.figure()  
hist = plt.hist(neg\_lr, bins=np.logspace(np.log10(0.03),np.log10(0.3), 20), density=True)  
plt.yscale("log")  
plt.xscale("log")  
plt.ylabel('Розп. ймов.')  
plt.xlabel('Щоденні негативні абсолютні прибутковості')  
  
plt.ylim([0.1, 110])  
plt.xlim([0.03, 0.3])  
plt.xticks([0.03, 0.04, 0.05, 0.06, 0.1, 0.2, 0.3], [0.03, 0.04, 0.05, 0.06, 0.1, 0.2, 0.3])  
plt.show();



Як ми бачимо, ця гістограма екстремальних абсолютних прибутків зменшується приблизно лінійно при логарифмічному масштабуванні. Щоразу, коли ви виявляєте пряму лінію на логарифмічному графіку, це вказує на так званий [степеневий розподіл](https://en.wikipedia.org/wiki/Power_law). Степеневе співвідношення ймовірності спостереження великих абсолютних логарифмічних прибутковостей задається у вигляді:

Константа на даний момент нас не надто турбує, вона просто гарантує, що ліва частина рівняння належним чином нормалізована, але показник представляє для найбільший інтерес, оскільки він говорить нам, як працює степеневий закон. Якщо ви знаєте частоту даних абсолютних логарифмічних прибутковостей, то степеневий закон підкаже вам, у скільки разів менш імовірними були б логарифмічні прибутковості подвоєного розміру:

або простіше:

Для всі прибутковості однаково ймовірні, для подвоєння прибутковостей робить їх у 4 рази менш імовірними. Оскільки це правило подвоєння вгору або вниз не залежить від значення самого , але працює для всіх значень . Степеневе співвідношення також називаються “безмасштабним” або [масштабо-інваріантним](https://en.wikipedia.org/wiki/Scale_invariance).

Як щодо показника для наших надзвичайно негативних логарифмічних прибутковостей S&P 500? Ми можемо легко оцінити його за даними, але є деякі тонкощі у правильному розбитті вищезазначеної гістограми, які можуть вплинути на нашу оцінку (див., наприклад, [White et al. (2008)](https://esajournals.onlinelibrary.wiley.com/doi/10.1890/07-1288.1)). Щоб обійти ці проблеми з розбиттям, ми можемо замість цього оцінити степенний показник [кумулятивної гістограми](https://en.wikipedia.org/wiki/Histogram#Cumulative_histogram) логарифмічних прибутковостей, а потім відняти одиницю від отриманого нахилу:

# підганяємо лінію регресії до кумулятивної гістограми (log10 для ймовірності) vs. лог-прибутковості (log10)  
sorted\_neg\_lr = np.sort(neg\_lr.values)  
cumulative\_probability = np.linspace(1, 0, len(sorted\_neg\_lr)+1)[:-1]  
  
x\_min = 0.03  
mask = sorted\_neg\_lr >= x\_min  
m, b = np.polyfit(np.log10(sorted\_neg\_lr[mask]), np.log10(cumulative\_probability[mask]), 1)  
alpha = -(m - 1)  
  
plt.figure()  
plt.plot(np.log10(sorted\_neg\_lr[mask]), np.log10(cumulative\_probability[mask]))  
plt.scatter(np.log10(sorted\_neg\_lr[mask]), np.log10(cumulative\_probability[mask]), label='sample data')  
  
x\_fit = np.linspace(np.log10(x\_min), np.log10(1.1\*np.max(neg\_lr)), 100)  
y\_fit = m\*x\_fit + b  
  
plt.plot(x\_fit, y\_fit, label='степенева підгонка')  
plt.title(f'розрахований степеневий показник $\\alpha={alpha:.2f}$')  
plt.xlabel('лог-прибутковості (log10-scaled)')  
plt.ylabel('кумулятивна гістограма (log10-scaled)')  
plt.show();



Виходячи з цієї оцінки, ми отримуємо степеневої показник, рівний . Якщо ми побудуємо гістограму наших надзвичайно негативних логарифмічних прибутковостей, то побачимо, що вона добре відображає зниження ймовірності, оскільки масштаб збоїв збільшується. Однак цей степеневий показник все ще недооцінює ймовірність настання чергового “Чорного понеділка”, оскільки лінія відповідності не ідеально підходить до цієї крайньої точки. Якби ми захотіли створити ще більш консервативну модель екстремальних подій, нам потрібно було б вручну ще більше зменшити значення , щоб врахувати більшу ймовірність настання чорних понеділків за рахунок втрати точності підгонки для менших збоїв. Це перший раз, коли ми можемо побачити, як одна точка даних впливає на наші рішення щодо моделювання. На даний момент ми будемо довіряти оцінці параметра і погодимося з оцінкою .

neg\_lr = -lr[lr < 0]  
  
plt.figure()  
hist = plt.hist(neg\_lr, bins=np.logspace(np.log10(0.03),np.log10(0.3), 20), density=True)  
plt.yscale("log")  
plt.xscale("log")  
plt.ylabel('Розп. ймов.')  
plt.xlabel('Щоденні негативні абсолютні прибутковості')  
plt.ylim([0.1, 110]);  
plt.xlim([0.03, 0.3])  
plt.xticks([0.03, 0.04, 0.05, 0.06, 0.1, 0.2, 0.3], [0.03, 0.04, 0.05, 0.06, 0.1, 0.2, 0.3]);  
  
x\_fit = np.logspace(np.log10(0.03),np.log10(0.3), 100)  
y\_fit = 0.00022\*(x\_fit\*\*-3.7)  
  
plt.plot(x\_fit, y\_fit, lw=3)  
plt.show();



Ми можемо використовувати цей степеневий закон, щоб екстраполювати, як часто ми очікуємо настання чорного понеділка (або “події з 23 сигмами”) у довгостроковій перспективі. Виходячи з наявних у нас даних, ми можемо оцінити, що ймовірність спостереження зниження на або гірше становить близько . Слідуючи нашій степеневій кривій, зниження чорного понеділка на , таким чином, відбувалося б із частотою . Так само часто, як зниження на , яке ми спостерігаємо приблизно раз на три місяці. Це означає, що, виходячи з нашої степеневої кривої, очікується, що чорні понеділки, подібні до 1987 року, траплятимуться приблизно раз на 450 років (див. код нижче).

# як часто ринок знижується більш ніж на -0.03  
(neg\_lr > 0.03).mean()

0.016091417910447763

# у середньому, як багато днів між цима трьома подіями  
1./(neg\_lr > 0.03).mean()

62.144927536231876

# у скільки разів менше відбувається подій "Чорного понеділка" в порівнянні зі зниженням на -0.03?  
(np.max(neg\_lr)/0.03)\*\*3.7

1845.1139014672306

np.max(neg\_lr)

0.2289972265656708

# скільки років пройшло між двома подіями "Чорного понеділка",  
# виходячи з припущення, що подія -0.03 відбувається один раз у 63 дня?  
(1./((1/62)\*1/1845))/252

453.92857142857144

Якби ми включили більше даних, що містять інші приклади екстремальних подій, наприклад, з [Великої депресії](https://en.wikipedia.org/wiki/Great_Depression), ця оцінка може стати ще меншою. Аналогічно, якби ми скоригували степеневий показник так, щоб він був більш консервативним, ніж передбачає наш простий метод оцінки, ми також отримали б коротший період. Хоча 460 років все ще є дуже великим періодом, це набагато більш реалістична оцінка порівняно з роками, які ми отримали, використовуючи розподіл Гауса. Однією з переваг степеневої моделі є її здатність до екстраполяції: використовуючи степеневу оцінку та частоту менших прибутковостей, щодо яких ми маємо достовірні дані, ми можемо оцінити частоту майбутніх серйозних збоїв, щодо яких ми ще не отримали жодних даних. Ще про одну перевагу оцінки ми розглянемо далі.

|  |
| --- |
| ПРИМІТКА |
| Не існує такого поняття, як “подія з 10 сигмами”! Справжня подія з 10 сигмами настільки неймовірно рідкісна, що пережити її протягом нашого життя — мізерно мало. Екстремальні події, безумовно, трапляються, але вони не можуть бути описані за допомогою “сигм”, їх потрібно враховувати і екстраполювати з використанням степеневих законів. Завжди будьте обережні, коли аналітики виправдовують себе за те, що вони не були готові до “події з 10 сигмами”, оскільки це вказує на те, що їх моделі ризиків вкрай недосконалі. |

### 10.1.7 Статистичні моменти під впливом важких хвостів

Чому все це так важливо? Степенневий показник насправді багато говорить нам про стабільність і збіжність [моментів розподілу ймовірностей](https://en.wikipedia.org/wiki/Moment_(mathematics)), і це, в свою чергу, має значення для деяких найбільш часто використовуваних моделей у фінансах. Перші чотири центральні моменти — це:

* [**Середнє**](https://en.wikipedia.org/wiki/Expected_value): очікуване значення розподілу
* [**Дисперсія**](https://en.wikipedia.org/wiki/Variance): Квадрат стандартного відхилення, який часто використовується для оцінки волатильності на основі розподілу прибутковості.
* [**Асиметрія**](https://en.wikipedia.org/wiki/Skewness): вимірює зміщення розподілу. Розподіл прибутковостей, як правило, має негативну асиметрію, оскільки збої є більш стрімкими, ніж підйоми.
* [**Ексцес**](https://en.wikipedia.org/wiki/Kurtosis): вимірює тяжкість хвостів розподілу. Розподіли прибутковостей зазвичай мають ексцес більший ніж у розподілі Гауса, тобто екстремальні події відбуваються частіше, ніж очікувалося в гаусовій моделі.

Ексцес Гаусового розподілу дорівнює 3 (використовуючи визначення Пірсона, з визначенням Фішера [надлишкового ексцесу](https://en.wikipedia.org/wiki/Kurtosis#Excess_kurtosis) це 0), але наша вибірка щоденних логарифмічних прибутковостей S&P 500 має ексцес , що знову вказує на те, що екстремальні події набагато більш вірогідні, що суперечить Гаусовому розподілу.

# обчислюємо ексцес лог-прибутковостей  
from scipy.stats import kurtosis  
  
kurtosis(lr, fisher=False, bias=False)

28.610931957080663

Тепер ми дійшли до того моменту, коли ми могли б запитати: наскільки сильно одна точка впливає на нашу оцінку ексцесу? Якщо ми перерахуємо ексцес всіх щоденних прибутків S&P 500 з 1950 року і виключимо тільки чорний понеділок 1987 року, ми отримаємо значення , майже половину від значення, яке ми отримуємо, коли використовуємо всі значення! Видалення подальших найгірших днів, звичайно, ще більше зменшує ексцес, але ефект значно менший.

# обчислюємо ексцес лог-прибутковостей після видалення Чорного понеділка 1987  
kurtosis(lr[lr > np.min(lr)], fisher=False, bias=False)

14.3083633314701

lr\_sorted = lr.sort\_values()  
kurt = [kurtosis(lr\_sorted.iloc[i:], fisher=False, bias=False) for i in range(10)]  
  
plt.figure()  
plt.plot(np.arange(10), kurt, c='C3', lw=2, zorder=2)  
plt.scatter(np.arange(10), kurt, s=120, lw=0.75, edgecolor='k', facecolor='C3', zorder=3)  
plt.grid()  
plt.xlabel('кіль-ть виключених найгірших днів')  
plt.ylabel('Ексцес щоденних лог-прибутковостей')  
plt.ylim([0, 32])  
plt.show();



Більш радикальний погляд на цю проблему досягається, якщо ми будуємо графік ексцесу в покроковому режимі, тобто для кожного дня ми будуємо розрахунковий ексцес, використовуючи всі минулі логарифмічні дані до цього моменту часу. Як ви можете бачити нижче, до “Чорного понеділка” ексцес ніколи не перевищував , проте навіть через 35 років після “Чорного понеділка” ексцес ще не “оговтався” від цієї події і залишається на позначці 30. Чи означає це, що ми повинні просто ігнорувати Чорний понеділок як викид і продовжувати використовувати нашу “чисту” оцінку в ? Ні, точно ні! Як ми побачимо незабаром, ми скоріше повинні запитати себе, чи є сенс у тому, щоб оцінювати ексцес!

kurt = [kurtosis(lr.iloc[:i], fisher=False, bias=False) for i in range(1000, len(lr))]

plt.figure()  
plt.plot(lr.index[1000:], kurt, c='C3', lw=3, zorder=2)  
plt.ylabel('Ексцес щоденних лог-прибутковостей')  
plt.show();



Той факт, що одна точка даних може настільки сильно змінити нашу оцінку ексцесу, і, схоже, вона більше не сходиться, коли додаються нові точки даних, є чітким свідченням того, що ексцес базового розподілу ймовірностей насправді нескінченний! Проблема полягає в тому, що для кінцевої вибірки даних (а всі набори даних скінченні) ми завжди зможемо оцінити скінчене значення ексцесу. scipy.stats.kurtosis не може повернути , він завжди повідомлятиме про скінченний ексцес *вибірки*. Але ця кінцева оцінка не допоможе нам нічого сказати про майбутню поведінку ринку, оскільки ексцес продовжуватиме переходити до ще більших значень до наступного чорного понеділка, іншого чорного вівторка … у більш-менш віддаленому майбутньому.

Однак опис частоти екстремальних подій згідно степеневого закону, який ми представили вище, може допомогти нам вирішити, чи є ексцес кінцевим, так що, переглянувши достатню кількість точок даних, ми зможемо вирішити, чи слід прийняти той факт, що ексцес не піддається оцінці і справді нескінченний. Щоб побачити це, нам потрібно ввести ще один розподіл ймовірностей, [t-розподіл Стьюдента](https://en.wikipedia.org/wiki/Student%27s_t-distribution). Це колоколообразний симетричний розподіл зі степеневими хвостами. Його додатковий параметр визначає показник степеневого закону . Нижче ми підганяємо t-розподіл до наших логарифмічних прибутковостей S&P 500, зберігаючи при цьому степеневий показник (відповідний ), який ми підганяли раніше:

from scipy.stats import t  
  
fit\_params = t.fit(lr, fix\_df=2.7)  
  
x\_fit = np.linspace(-0.25, 0.125, 1000)  
y\_fit\_t = t(\*fit\_params).pdf(x\_fit)

plt.figure()  
plt.hist(lr, 100, density=True)  
plt.plot(x\_fit, y\_fit\_t, lw=3, c='C3', alpha=0.8, label="Підігнаний t-розподіл")  
plt.yscale('log')  
plt.ylabel('Розп. ймов.')  
plt.xlabel('Щоденні лог-прибутковості')  
plt.xlim([-0.25, 0.125])  
plt.ylim([0.0011, 99])  
plt.legend()  
  
plt.title(f'Підігнаний степеневий показник: $\\alpha = {(fit\_params[0]+1):.2f}$')  
plt.show();



Звичайно, можна поставити під сумнів, чи дійсно t-розподіл Стьюдента добре відповідає нашим логарифмічним прибутковостям, оскільки він переоцінює частоту абсолютних прибутковостей і все ще недооцінює ймовірність настання Чорного понеділка. На даний момент ми ігноруємо ці деталі та зосереджуємось на ексцесі t-розподілу Стьюдента та зв’язку зі степеневим показником :

* для або
* для або
* інакше невизначено

Це підтверджує наше початкове припущення про те, що при степеневому показнику, рівному , ексцес дійсно нескінченний, і немає сенсу оцінювати його по скінченій вибірці, оскільки оціночне значення буде зберігатися тільки до наступної екстремальної події, яка підштовхне його ще вище. Ми можемо легко змоделювати цей ефект, витягуючи випадкові вибірки з t-розподілів Стьюдента з різними степеневими показниками, які відповідають трьом режимам, зазначеним вище. Нижче ми витягуємо 100 000 випадкових значень з t-розподілів з , , , а потім обчислюємо ексцес поетапно, як і раніше, щоб побачити, як еволюціонує передбачуваний ексцес вибірки з часом у трьох різних випадках:

np.random.seed(1)  
samples\_alpha\_20 = t(1.0, 0., 1.).rvs(100000)  
samples\_alpha\_37 = t(2.7, 0., 1.).rvs(100000)  
samples\_alpha\_60 = t(5.0, 0., 1.).rvs(100000)

m = np.arange(1000, 100000, 50)  
kurt\_alpha\_20 = [kurtosis(samples\_alpha\_20[:i], fisher=False, bias=False) for i in m]  
kurt\_alpha\_37 = [kurtosis(samples\_alpha\_37[:i], fisher=False, bias=False) for i in m]  
kurt\_alpha\_60 = [kurtosis(samples\_alpha\_60[:i], fisher=False, bias=False) for i in m]

plt.figure()  
  
plt.subplot(311)  
plt.plot(m, kurt\_alpha\_60, c='C2', lw=3, zorder=2)  
plt.axhline(6/(5-4) + 3, lw=1, ls='--', c='k')  
plt.ylabel('Ексцес')  
plt.xlabel('Кількість значень')  
plt.title('Степеневий показник $\\alpha=6$')  
  
plt.subplot(312)  
plt.plot(m, kurt\_alpha\_37, c='C1', lw=3, zorder=2)  
plt.ylabel('Ексцес')  
plt.xlabel('Кількість значень')  
plt.title('Степеневий показник $\\alpha=3.7$')  
  
plt.subplot(313)  
plt.plot(m, kurt\_alpha\_20, c='C3', lw=3, zorder=2)  
plt.ylabel('Ексцес')  
plt.xlabel('Кількість значень')  
plt.title('Степеневий показник $\\alpha=2$')  
  
plt.tight\_layout()  
plt.show();



У випадку () ексцес скінченний і повинен приймати значення . Хоча ми бачимо деякі коливання, оціночне значення ексцесу повільно наближається до справжнього значення. Для прикладу, який відповідає степеневому показнику індексу S&P 500 з , моделювання показує таку ж поведінку, що і для реальних даних: збіжність не може бути виявлена, скоріше кожна екстремальна подія збільшує ексцес; при нескінченних вибірках ми би досягли нескінченного ексцесу. У третьому випадку з ексцес t-розподілу Стьюдента навіть не визначений належним чином, і ми можемо побачити інший тип поведінки моделювання: зараз більшість точок даних збільшують оцінку ексцесу, а не лише кілька окремих екстремальних зразків. При такому повільно спадаючому статечному законі екстремальні значення зустрічаються повсюдно, що призводить до збільшення ексцесу при моделюванні.

|  |
| --- |
| Примітка |
| Навіть якщо певні властивості розподілу ймовірностей нескінченні, ми завжди знайдемо кінцеві вибіркові оцінки при перегляді даного набору даних просто тому, що набір даних містить лише кінцеву кількість точок. Однак, якщо теорія степеневого закону чітко говорить нам, що певна властивість нескінченна, ми не повинні ніде використовувати кінцеві оцінки і ми не повинні використовувати моделі, які вимагають, щоб ця властивість була кінцевою! Якщо ми все одно це зробимо, це в кращому випадку буде працювати лише до наступної екстремальної події, яка анулює наші минулі оцінки і — у випадку фінансових моделей — може збанкрутувати нас! |

### 10.1.8 Як ексцес дестабілізує розрахунки волатильності

Оскільки ексцес збільшується — або стає нескінченним — це також впливає на нижчі моменти, найголовніше, на дисперсію. Квадратний корінь дисперсії — це стандартне відхилення, або [волатильність](https://en.wikipedia.org/wiki/Volatility_(finance)), як це називається у фінансах. Волатильність є розповсюдженним параметром при оптимізації портфеля і практично в кожній фінансовій моделі, оскільки в кінцевому підсумку вона стала найбільш часто використовуваним показником “ризику”. Щоб побачити, як ексцес впливає на оцінку волатильності, ми проводимо простий експеримент [Монте-Карло](https://en.wikipedia.org/wiki/Monte_Carlo_method): ми витягуємо щоденну прибутковість за 50 років з розподілу ймовірностей за нашим вибором (ми будемо використовувати Гаусовий і t-розподіл Стьюдента) і обчислюємо оцінку волатильності, використовуючи вибіркове стандартне відхилення. Ми робимо це не тільки один раз, але і 10000 разів, кожен раз, коли ми розраховуємо нову гіпотетичну щоденну прибутковість за 50 років і обчислюємо відповідну оцінку волатильності. Звичайно, не всі ці оцінки будуть однаковими, але коливатимуться навколо справжньої волатильності, оскільки ми маємо лише кінцеву кількість даних. Оскільки ми знаємо справжній розподіл ймовірностей, який ми використовуємо для моделювання, ми також знаємо справжню волатильність. Щоб побачити, наскільки точно ми можемо оцінити волатильність на основі даних за 50 років, ми обчислюємо відносну похибку між оцінками волатильності та справжньою волатильністю.

n\_sim = 10000  
  
true\_var = 1.  
true\_vol = np.sqrt(true\_var)  
  
rand\_vol = np.array([np.std(norm(0., 1.).rvs(50\*252), ddof=1) for i in tqdm(range(n\_sim))])  
  
100\*np.mean(np.abs(rand\_vol - true\_vol))/true\_vol

100%|██████████| 10000/10000 [00:08<00:00, 1125.24it/s]

0.5039568642168273

n\_sim = 10000  
  
alpha = 6.0  
nu = alpha - 1  
  
true\_var = nu/(nu-2)  
true\_vol = np.sqrt(true\_var)  
  
rand\_vol = np.array([np.std(t(nu, 0., 1.).rvs(50\*252), ddof=1) for i in tqdm(range(n\_sim))])  
  
100\*np.mean(np.abs(rand\_vol - true\_vol))/true\_vol

100%|██████████| 10000/10000 [00:13<00:00, 728.17it/s]

0.9616583563963519

n\_sim = 10000  
  
alpha = 4.0  
nu = alpha - 1  
  
true\_var = nu/(nu-2)  
true\_vol = np.sqrt(true\_var)  
  
rand\_vol = np.array([np.std(t(nu, 0., 1.).rvs(50\*252), ddof=1) for i in tqdm(range(n\_sim))])  
  
100\*np.mean(np.abs(rand\_vol - true\_vol))/true\_vol

100%|██████████| 10000/10000 [00:14<00:00, 700.76it/s]

3.8082666425995932

n\_sim = 10000  
  
alpha = 3.7  
nu = alpha - 1  
  
true\_var = nu/(nu-2)  
true\_vol = np.sqrt(true\_var)  
  
rand\_vol = np.array([np.std(t(nu, 0., 1.).rvs(50\*252), ddof=1) for i in tqdm(range(n\_sim))])  
  
100\*np.mean(np.abs(rand\_vol - true\_vol))/true\_vol

100%|██████████| 10000/10000 [00:14<00:00, 709.36it/s]

6.115969986022528

n\_sim = 10000  
  
alpha = 3.1  
nu = alpha - 1  
  
true\_var = nu/(nu-2)  
true\_vol = np.sqrt(true\_var)  
  
rand\_vol = np.array([np.std(t(nu, 0., 1.).rvs(50\*252), ddof=1) for i in tqdm(range(n\_sim))])  
  
100\*np.mean(np.abs(rand\_vol - true\_vol))/true\_vol

100%|██████████| 10000/10000 [00:13<00:00, 718.23it/s]

38.47707859009605

Ми отримуємо наступні відносні похибки для різних розподілів та їх параметрів: - Гаусовий розподіл; відн. похиб. - t-розподіл Стьюдента з (); відн. похиб. - t-розподіл Стьюдента з (); відн. похиб. - t-розподіл Стьюдента з (); відн. похиб.

При степеневому показнику (що відповідає ; як ми підрахували для S&P 500, щоб пояснити Чорний понеділок), щоденна прибутковість за 50 років все ще призводить до помилки оцінки волатильності більш ніж на ! Зверніть увагу, що ця велика помилка оцінки зростатиме ще швидше, коли вона наближається до , оскільки дисперсія стає нескінченною для t-розподілу Стьюдента при (). Виходячи з t-розподілу прибутковостей S&P 500, похибка оцінки волатильності зросла в 12 разів (!) порівняно з тим, що ми очікували б, припускаючи, що прибутковості розподілені за Гаусом.

|  |
| --- |
| Примітка |
| Чим важчі хвости розподілу прибутковостей, тим більше точок даних необхідно для досягнення необхідної достовірності оцінок певних параметрів. Пам’ятайте про це щоразу, коли ви намагаєтеся оцінити ризик інвестування у відносно новий фінансовий актив, який має дані лише за кілька років. |

### 10.1.9 Наслідки для моделювання та прогнозування

Той факт, що більш високі моменти розподілу прибутковості можуть бути нескінченними і що оцінка нижчих моментів, таких як дисперсія, дестабілізується за наявності “товстих хвостів”, має суттєві наслідки для деяких найбільш використовуваних моделей у фінансах. Ми коротко розглянемо дві з них, щоб проілюструвати проблему.

### 10.1.10 Марковіц і стабільність волатильності

У 1954 році [Гаррі Марковіц](https://en.wikipedia.org/wiki/Harry_Markowitz) отримав ступінь доктора економічних наук за роботу з теорії фінансових портфелів. Його робота стала наріжним каменем [сучасної портфельної теорії](https://en.wikipedia.org/wiki/Modern_portfolio_theory) і тоді це було настільки новим, що під час захисту докторської дисертації Мілтон Фрідман стверджував, що ця робота насправді не стосується галузі економіки (як пояснив Марковіц у своїй [Нобелівській лекції](http://nobelprize.org/nobel_prizes/economics/laureates/1990/markowitz-lecture.html) у 1990 році). Марковіц стверджував, що при заданому рівні волатильності існує набір вагових коефіцієнтів розподілу активів у портфелі, які дають максимальну надлишкову прибутковість портфеля (порівняно з безризиковим активом). Якщо ви переглянете діапазон значень волатильності та обчислите максимальну віддачу портфеля, отримавши оптимальні ваги розподілу, Ви отримаєте набір *ефективних* портфелів, який також називають [межею ефективності](https://en.wikipedia.org/wiki/Efficient_frontier). Одна точка на цій межі в багатьох випадках представляє особливий інтерес, — це точка максимуму [коефіцієнта Шарпа](https://en.wikipedia.org/wiki/Sharpe_ratio).

Коефіцієнт Шарпа — це часто критикуваний, але також часто використовуваний показник для оцінки ефективності портфеля або фінансових активів загалом, таких як гедж-фонди або ETF. Він визначається як:

Тут позначає річну прибутковість портфеля, позначає річну безризикову прибутковість, а позначає річну волатильність портфеля. Припускаючи нульову безризикову ставку (), коефіцієнт Шарпа можна інтерпретувати як [співвідношення сигнал/шум](https://en.wikipedia.org/wiki/Signal-to-noise_ratio), відомий у фізиці та інженерії: максимізуючи коефіцієнт Шарпа, ми знаходимо найкращий компромісне значення, що максимізує наші прибутковості (сигнал) та мінімізує волатильності (шум).

Якщо наш портфель складається з фінансових активів з очікуваною прибутковістю , з коваріаційною матрицею , яка містить інформацію про волатильність і кореляції між активами, і з вектором ваги розподілу (нормалізовано до одиниці: ), тоді коефіцієнт Шарпа у контексті сучасної теорії портфеля можна обчислити наступним чином:

Зверніть увагу, що з цього моменту ми завжди припускаємо, що .

Щоб максимізувати прибутковість відносно волатильності та отримати портфель з максимальним коефіцієнтом Шарпа, ми змінюємо ваги розподілу, поки не знайдемо набір ваг з максимальним коефіцієнтом Шарпа:

Основне припущення, що лежить в основі сучасної теорії портфеля, полягає в тому, що об’єднаний розподіл логарифмічних прибутковостей всіх активів у портфелі може бути повністю охарактеризований мультиваріативним розподілом Гауса. Після попередніх частин цієї лабораторної ми дуже добре розуміємо цю помилку, і сучасна теорія портфеля по праву зазнала жорсткої критики за відсутність врахування “важких хвостів”, наприклад, з боку Нассіма Талеба:

|  |
| --- |
| Нассім Талеб у *Чорний лебідь: Про (не)ймовірне в реальному житті* (ст. 277) |
| Після краху фондового ринку (у 1987 році) вони нагородили двох теоретиків, Гаррі Марковіца та Вільяма Шарпа, які побудували прекрасні платонівські моделі на гаусовій основі, сприяючи тому, що називається сучасною теорією портфеля. Простіше кажучи, якщо ви видалите їх гаусові припущення і розглядаєте ціни як масштабо-інваріантні, у вас залишиться гаряче повітря. Нобелівський комітет міг би протестувати моделі Шарпа і Марковіца — вони працюють як шарлатанські ліки, що продаються в Інтернеті, — але ніхто в Стокгольмі, схоже, про це не подумав. |

Тут ми просто хочемо проілюструвати наслідки припущення гаусовости, змоделювавши прибутковість двох гіпотетичних акцій і спробувавши знайти ваги розподілу по максимуму Шарпа. Як ми побачимо, великі хвости в модельованих розподілах прибутковостей вносять істотну невизначеність в розрахункові ваги розподілу — прямий наслідок нестабільності дисперсії.

Спочатку ми генеруємо випадкові числа з Гаусового розподілу, достатнього для моделювання щоденної прибутковості двох гіпотетичних акцій за 10 років. Ми присвоюємо їм різну позитивну очікувану прибутковість і волатильність. Для простоти ми припускаємо, що кореляція між двома активами є нульовою:

np.random.seed(1234)  
n\_years = 10  
  
log\_ret\_1 = 0.0025\*norm.rvs(size=n\_years\*252) + 0.0005  
log\_ret\_2 = 0.0050\*norm.rvs(size=n\_years\*252) + 0.0010  
  
plt.figure()  
plt.plot(np.exp(np.cumsum(log\_ret\_1)), label='Акція 1')  
plt.plot(np.exp(np.cumsum(log\_ret\_2)), label='Акція 2')  
  
plt.ylabel('Ціна акції')  
plt.xlabel('Торговий день')  
plt.legend()  
plt.yscale('log')  
plt.show();



Звичайно, насправді ми не знаємо *справжньої* очікуваної прибутковості або значень волатильності кореляцій між активами. Таким чином, ми повинні оцінити очікувану прибутковість за середнім значенням вибірки та матрицю коваріації за матрицею коваріації вибірки. Зверніть увагу, що існують більш складні способи оцінки цих параметрів на основі історичних даних, таких як [оцінки усадки](http://www.ledoit.net/honey.pdf) для коваріаційної матриці або [моделі розподілу Блека-Літтермана](https://sci-hub.mksa.top/10.3905/jfi.1991.408013). Ці методи, безумовно, покращують вибіркові оцінки, але загальний недолік сучасної теорії портфеля через наявність “важких хвостів” залишається. Щоб вивчити та протестувати різні методи оптимізації портфеля, рекомендуємо розглянути бібліотеку [PyPortfolioOpt](https://pyportfolioopt.readthedocs.io/en/latest/index.html), що представляє собою добре задокументований пакет Python, який реалізує як основні, так і складні методи оптимізації портфеля та дозволяє легко тестувати ці методи на реальних даних. Поки ми дотримуємося вибіркових оцінок (і використовуємо коефіцієнт 252 — середню кількість торгових днів на рік для отримання очікуваної прибутковості в річному обчисленні і коваріації за щоденними оцінками):

exp\_returns = np.array([np.mean(log\_ret\_1), np.mean(log\_ret\_2)])\*252  
cov\_matrix = np.cov([log\_ret\_1, log\_ret\_2])\*252

Далі ми визначаємо функцію, яка приймає ваги розподілу кандидатів, розрахункову очікувану прибутковість і коваріаційну матрицю і виводить коефіцієнт Шарпа портфеля:

def sharpe(weights, exp\_returns, cov\_matrix):  
 return weights.dot(exp\_returns) / np.sqrt(weights.dot(cov\_matrix).dot(weights.T))  
  
sharpe(np.array([0.5, 0.5]), exp\_returns, cov\_matrix)

4.810679451759804

Замість того, щоб використовувати реальний оптимізатор для пошуку ваг розподілу, які максимізують коефіцієнт Шарпа, ми просто переглядаємо всі можливі комбінації ваг з інтервалом . Звичайно, це розумний підхід лише для двох активів, і він швидко стає нерозв’язним, використовуючи більше двох активів.

candidate\_weights = np.array([np.linspace(0, 1, 101), 1. - np.linspace(0, 1, 101)]).T  
candidate\_weights[:5]

array([[0. , 1. ],  
 [0.01, 0.99],  
 [0.02, 0.98],  
 [0.03, 0.97],  
 [0.04, 0.96]])

Тепер ми можемо перебрати всі наші ваги-кандидати, обчислити коефіцієнт Шарпа портфеля для всіх з них, а потім вибрати оптимальний набір ваг.

candidate\_sharpe = [sharpe(weights, exp\_returns, cov\_matrix) for weights in candidate\_weights]  
opt\_weights = candidate\_weights[np.argmax(candidate\_sharpe)]  
opt\_weights

array([0.7, 0.3])

Щоб отримати уявлення про те, наскільки точно ми можемо оцінити ваги розподілу на основі гіпотетичної щоденної прибутковості за 10 років, ми зараз повторюємо цей експеримент, повторно генеруємо нові випадкові прибутковості, повторно оцінюємо очікувані прибутковості та матрицю коваріації та знаходимо оптимальні ваги. Ми робимо це 10000 разів. Зверніть увагу, що ми реєструємо лише першу вагу розподілу для кожного запуску без втрати інформації, оскільки друга вага безпосередньо випливає з нормалізації .

np.random.seed(1234)  
opt\_weight\_gaussian = []  
  
for i in tqdm(range(10000)):  
 log\_ret\_1 = 0.0025\*norm.rvs(size=n\_years\*252) + 0.0005  
 log\_ret\_2 = 0.0050\*norm.rvs(size=n\_years\*252) + 0.0010  
  
 exp\_returns = [np.mean(log\_ret\_1), np.mean(log\_ret\_2)]  
 cov\_matrix = np.cov([log\_ret\_1, log\_ret\_2])  
  
 candidate\_sharpe = [sharpe(weights, exp\_returns, cov\_matrix) for weights in candidate\_weights]  
 opt\_weights = candidate\_weights[np.argmax(candidate\_sharpe)]  
  
 opt\_weight\_gaussian.append(opt\_weights[0])

100%|██████████| 10000/10000 [00:07<00:00, 1358.21it/s]

Нарешті, ми можемо побудувати гістограму всіх 10000 значень, які ми отримуємо для

plt.figure()  
plt.hist(opt\_weight\_gaussian, 10, density=True);  
plt.xlim([0, 1])  
plt.xlabel('Вага $w\_1$ для портфеля з максимальним коеф. Шарпа')  
plt.ylabel('Розп. ймов.')  
plt.show();



Як ми бачимо, розподіл для першої акції коливається приблизно від до на основі історичних даних за 10 років. Ми можемо додатково оцінити це кількісно, обчисливши 5-й і 95-й процентилі всіх значень . Таким чином, з упевненістю в ми отримаємо розподіл між і :

np.percentile(opt\_weight\_gaussian, 5), np.percentile(opt\_weight\_gaussian, 95)

(0.61, 0.72)

Тепер давайте повторимо цей експеримент, але цього разу ми замінимо розподіл Гауса на розподіл Стьюдента зі степеневим показником (що відповідає кількості ступенів свободи ). Ми залишаємо очікувану прибутковість і волатильність (виміряні стандартним відхиленням розподілу) точно такими ж. Зверніть увагу, що стандартне відхилення стандартного t-розподілу визначається як в нашому випадку, тому ми повинні внести поправку на цей коефіцієнт маштабування при вилученні випадкових значень зі стандартного t-розподілу. Нижче ви можете побачити пряме порівняння розподілів Гауса, використаних у наведеному вище моделюванні, та t-розподілів із відповідним середнім значенням та стандартним відхиленням.

scale\_correction = np.sqrt(2.7/(2.7-2.))

x = np.linspace(-0.025, 0.025, 10000)  
  
y1\_norm = norm(loc=0.0005, scale=0.0025).pdf(x)  
y1\_t = t(loc=0.0005, scale=0.0025/scale\_correction, df=2.7).pdf(x)  
  
y2\_norm = norm(loc=0.001, scale=0.005).pdf(x)  
y2\_t = t(loc=0.001, scale=0.005/scale\_correction, df=2.7).pdf(x)  
  
  
plt.figure()  
  
plt.subplot(121)  
  
plt.fill\_between(x, 0., y1\_norm, facecolor='C0', alpha=0.7)  
plt.fill\_between(x, 0., y1\_t, facecolor='C3', alpha=0.5)  
plt.ylim([0, 300])  
plt.ylabel('Розп. ймов.')  
plt.title('Акція 1')  
  
plt.subplot(122)  
plt.fill\_between(x, 0., y2\_norm, facecolor='C0', alpha=0.7, label='Гаус')  
plt.fill\_between(x, 0., y2\_t, facecolor='C3', alpha=0.5, label="Стьюдент")  
plt.ylim([0, 300])  
plt.ylabel('Розп. ймов.')  
plt.title('Акція 2')  
plt.legend()  
  
plt.tight\_layout()  
plt.show();



Як ми можемо легко бачити, узгоджені t-розподіли Стьюдента насправді дають менші значення прибутковостей. Однак вони також дають набагато більш екстремальні значення, але ми не можемо легко побачити це, оскільки значення щільності ймовірності в хвостах дуже малі! Якщо ми перейдемо до логарифмічного масштабування вісі Oy, картина стане більш чіткою:

plt.figure()  
  
plt.subplot(121)  
  
plt.plot(x, y1\_norm, c='C0', lw=3)  
plt.plot(x, y1\_t, c='C3', lw=3)  
plt.ylabel('Розп. ймов.')  
plt.title('Акція 1')  
plt.yscale('log')  
  
plt.subplot(122)  
plt.plot(x, y2\_norm, c='C0', lw=3, label='Гаус')  
plt.plot(x, y2\_t, c='C3', lw=3, label="Стьюдент")  
plt.ylabel('Розп. ймов.')  
plt.title('Акція 2')  
plt.yscale('log')  
plt.legend()  
  
plt.tight\_layout()  
plt.show();



Тепер ми чітко бачимо, що, наприклад, ймовірність щоденних логарифмічних прибутковостей для акції 1 в разів вища за умови t-розподілу порівняно з відповідним розподілом Гауса! Завжди дивіться на розподіли з логарифмічною шкалою щільності ймовірності, щоб отримати чітке уявлення про хвости. Використовуючи наші узгоджені t-розподіли, ми діємо, як і раніше, і моделюємо гіпотетичні 10-річні історичні показники для наших двох некорельованих акцій. Нижче ви можете побачити приклад. Тільки при уважному розгляді ви можете помітити, що ці криві сукупної прибутковості насправді трохи більш нерівні в порівнянні з тими, які були побудовані вище з гаусовими прибутковостями. Але різниця невловима, якщо дивитися на таку “наближену” діаграму десятирічного періоду:

np.random.seed(1234)  
n\_years = 10  
  
log\_ret\_1 = (0.0025/scale\_correction)\*t.rvs(df=2.7, size=n\_years\*252) + 0.0005  
log\_ret\_2 = (0.0050/scale\_correction)\*t.rvs(df=2.7, size=n\_years\*252) + 0.0010  
  
plt.figure()  
plt.plot(np.exp(np.cumsum(log\_ret\_1)), label='Акція 1')  
plt.plot(np.exp(np.cumsum(log\_ret\_2)), label='Акція 2')  
  
plt.ylabel('Ціна акції')  
plt.xlabel('Торговий день')  
plt.legend()  
plt.yscale('log')  
plt.show();



Нарешті, щоб візуалізувати вплив t-розподілу на оцінку ваги розподілу, ми повторюємо наш експеримент Монте-Карло з моделювання 10000 різних 10-річних історичних значень та оцінюємо ваги розподілу в кожному з цих 10000 випадків. Потім ми будуємо графік розподілу і порівнюємо цей розподіл з розподілом, отриманим в результаті моделювання на основі Гауса.

np.random.seed(1234)  
opt\_weight\_t = []  
  
for i in tqdm(range(10000)):  
 log\_ret\_1 = (0.0025/scale\_correction)\*t.rvs(df=2.7, size=n\_years\*252) + 0.0005  
 log\_ret\_2 = (0.0050/scale\_correction)\*t.rvs(df=2.7, size=n\_years\*252) + 0.0010  
  
 exp\_returns = [np.mean(log\_ret\_1), np.mean(log\_ret\_2)]  
 cov\_matrix = np.cov([log\_ret\_1, log\_ret\_2])  
  
 candidate\_sharpe = [sharpe(weights, exp\_returns, cov\_matrix) for weights in candidate\_weights]  
 opt\_weights = candidate\_weights[np.argmax(candidate\_sharpe)]  
  
 opt\_weight\_t.append(opt\_weights[0])

100%|██████████| 10000/10000 [00:09<00:00, 1037.43it/s]

plt.figure()  
plt.hist(opt\_weight\_gaussian, 10, alpha=0.8, density=True, label='Гаусові прибутковості')  
plt.hist(opt\_weight\_t, 20, alpha=0.8, density=True, label="Прибутковості Стьюдента")  
plt.xlim([0, 1])  
plt.legend()  
plt.xlabel('Вага $w\_1$ для портфеля з максимальним коеф. Шарпа')  
plt.ylabel('Розп. ймов.')  
plt.show();



np.percentile(opt\_weight\_t, 5), np.percentile(opt\_weight\_t, 95)

(0.54, 0.77)

Як ми можемо бачити, оптимальна вага розподілу тепер коливається між і , що приблизно в 2 рази перевищує ширину надійного інтервалу для оптимізації портфеля, яка дотримувалася припущення про гаусовість! Навіть при узгодженому стандартному відхиленні гаусового і t-розподілу, нескінченний ексцес t-розподілу дестабілізує вибіркову дисперсію наших змодельованих прибутковостей і, таким чином, знижує точність наших оціночних ваг розподілу. Хоча цей приклад, безумовно, не розкриває всіх слабкостей сучасної теорії портфеля, він допомагає нам побачити, як досить абстрактні ефекти, такі як нестабільність дисперсії через надмірний ексцес, в кінцевому результаті псують широко використовувані фінансові моделі. Використання модельованої прибутковості, заснованої на гаусовому розподілі порівняно з розподілом з великим хвостом, може допомогти виявити обмеження фінансових моделей або знайти мінімальний обсяг даних, необхідний для досягнення певної достовірності, перш ніж нова модель буде фактично запущена у виробництво!

|  |
| --- |
| Примітка |
| Важкі хвости розподілу прибутковостей можуть сильно вплинути на результати усталених математичних моделей у фінансах, або збільшуючи невизначеність параметрів (як у сучасній теорії портфеля), або приводячи до збоїв моделей поза вибірки через нескінченно великі моменти. У фінансах краще екстраполювати частоту екстремальних подій і готуватися до них за допомогою конкретних інструментів геджування, ніж намагатися передбачити майбутню прибутковість за допомогою моделей часових рядів! |

### 10.1.11 Розподіл Леві

Альфа-стабільний розподіл є узагальненням розподілу Гауса, яке цінується за те, що воно тягне за собою жирні хвости. З цієї причини він широко використовується при обробці сигналів, наприклад, у медицині чи фінансах.

Загальний клас стабільних розподілів був введений і отримав цю назву французьким математиком Полем Леві на початку 1920-х років.

Раніше ця тема привертала лише помірну увагу провідних експертів, хоча були і ентузіасти, з яких можна згадати російського математика Олександра Яковича Хінчіна. Натхненням для Леві стало бажання узагальнити відому центральну граничну теорему, згідно з якою будь-який розподіл ймовірностей з кінцевою дисперсією збігається до гауссового розподілу.

Стабільні розподіли мають три виняткові властивості, які можна коротко підсумувати, заявляючи, що вони:

* інваріантні при додаванні;
* мають власну область збіжності;
* дозволяють канонічну форму характеристичної функції.

#### 10.1.11.1 Інваріантність при додавані

*Випадкова величина підпорядковується стабільному розподілу якщо для будь-якого існують додатнє значення та дійсне значення такі, що*

*де характеризуються як незалежні, ідентично розподілені випадкові величини.* Також позначає *рівність розподілів*, тобто, випадкові величини з обох сторін мають однаковий розподіл ймовірностей.

Загалом, сума незалежних, ідентично розподілених випадкових величин результує у випадкову величину з іншим розподілом. Однак, для випадкових величин, що характеризуються *стабільним* розподілом, сума ідентично розподілених випадкових величин до величини такого самого розподілу. У цьому випадку результуюча випадкова величина (розподіл) може відрізнятися від попередніх величин характерним масштабом () та зміщенням (). Якщо , розподіл називається *строго стабільним*.

Відомо, що нормована константа має вид

Параметр має назву *характеристична експонента* або *індекс стабільності* стабільного розподілу.

Попередня теорема має альтернативну версію, що включає в суму лише дві випадкові величини. *Випадкова величина підпорядковується стабільному розподілу, якщо для будь-яких позитивних значень та існує позитивне число та дійсне число такі, що*

*де та незалежні копії* . Тоді існує значення при яких значення задовільняє рівність .

Для строго стабільних розподілів . Це означає, що всі лінійні комбінації випадкових незалежних, ідентично розподілених випадкових величин, що підкоряються строго стабільному розподілу, результують у випадкову величину з одним і тим же типом розподілу.

Стабільний розподіл вважається *симетричним*, якщо випадкова величина має такий самий тип розподілу. *Симетричний* стабільний розподіл обов’язково *строго стабільний*.

Оскільки аналітичний вираз функції щільності ймовірностей для стабільного розподілу невідома, за винятком кількох членів стабільного сімейства, більшість традиційних методів математичної статистики не можуть бути використані. Відбовідними вийнятками є

* **Гаусовий розподіл** . Гаусовий розподіл є спеціальним випадком стабільного розподілу при так що , де позначає середнє значення нормального розподілу, а — це стандартне відхилення. Функція щільності ймовірностей має вид
* **Розподіл Коші**. Розподіл Коші — це ще одне представлення стабільного розподілу при та такими, що , де — це параметр масштабування, а — це параметр зсуву розподілу Коші. Функція щільності ймовірностей представлена як:
* **Розподіл Леві** також є вийнятком із класу стабільних розподілів, де і . Іншими словами, . Функція щільності ймовірностей має вид

Якщо , тоді для

де позначає кумулятивну функцію нормального розподілу.

#### 10.1.11.2 Область збіжності

Інше (еквівалентне) визначення стверджує, що стабільні розподіли — це єдині розподіли, які можна отримати при границі нормалізованих сум незалежних, ідентично розподілених випадкових величин. Кажуть, що випадкова величина має область збіжності, тобто якщо існує послідовність незалежних, ідентично розподілених випадкових величин і послідовності позитивних чисел і дійсних чисел таких, що

Коли це гаусова випадкова величина, а є незалежними, ідентично розподіленими випадковими величинами з визначенною дисперсією, тоді рівняння вище є твердженням звичайної *центральної граничної теореми*. Область збіжності вважається *нормальною* коли .

#### 10.1.11.3 Канонічні представлення характеристичної функції

Чотири параметри використовуються для опису випадкової величини, що слідує за стабільним розподілом: . Параметр — це той, який нас найбільше зацікавить. Цей параметр визначає товщину хвостів. Параметр є параметром асиметрії. Останні два параметри позначають розташування і масштаб розподілу. Альфа-стабільний розподіл немає жодного аналітичного виразу для щільності ймовірності , але ми можемо охарактеризувати його характеристичною функцією:

Ми могли б використовувати перетворення Фур’є, щоб отримати функцію щільності розподілу ймовірностей з характеристичної функції:

Але наведена вище параметризація не є повністю задовільною, оскільки функція щільності розподілу ймовірностей не є безперервною, зокрема, при . Дійсно, коли , щільність розподілу зміщується вправо, коли і вліво, коли , зі зсувом в сторону (відповідно ), коли прагне до 1. Таким чином, для прикладного аналізу даних та інтерпретації коефіцієнтів слід уникати такої параметризації.

Існує багато параметризацій для стабільних законів, і ці різні параметризації викликали велику плутанину. Різноманітність параметризацій обумовлена поєднанням історичної еволюції плюс численними проблемами, які були проаналізовані за допомогою спеціалізованих форм стабільного розподілу. Є вагомі причини використовувати різні параметризації в різних ситуаціях. Якщо в пріорітеті чисельні розрахунки, робота із даними, то краще використовувати одну параметризацію. Якщо бажані прості алгебраїчні властивості розподілу або аналітичні властивості строго стійких законів, то краще розглянути декілька параметризацій. Нолан запропонував використовувати параметризацію Золотарьова (M), яку також часто позначають як . Характеристична функція, що відповідає , дорівнює:

Ця альтернативна параметризація недалека від зазначеної напочатку. Єдина відмінність стосується параметра , який у даній параметризації коригує зсув для значень близьких до 1:

#### 10.1.11.4 Метод розрахунку параметрів -стабільного розподілу

Числені методи, такі як метод Маккаллоха, заснований на квантилях, і метод оцінки максимальної правдоподібності були розроблені в результаті відсутності аналітичних рішень. Припустимо, що вектор, що складається з незалежних ідентично розподілених випадкових величин із розподілу Парето, і також . Визначивши , Митник, Доганоглу та Ченайо розробили алогритм максимальної правдоподібності і показали, що можна розрахувати, максимізуючи функцію логарифмічної правдоподібності

Дюмушель застосував метод максимальної правдоподібності до стабільного розподілу і визначив функцію правдоподібності наступним чином:

де опираючись на для розміру вибірки .

## 10.2 Хід роботи

### 10.2.1 Зчитування з Yahoo Finance

# встановлення назви індексу  
symbol = "^BSESN"   
  
# встановлення діапазону з яким будемо працювати  
start = "1980-01-01"  
end = "2022-11-07"  
  
# завантаження даних з Yahoo  
data = yf.download(symbol, start, end)  
time\_ser = data['Adj Close'].copy()  
  
# підпис по вісі Ох   
xlabel = 'time, days'  
  
# підпис по вісі Оу  
ylabel = symbol   
  
# збереження результату в текстовий документ   
np.savetxt(f'{symbol}\_initial\_time\_series.txt', time\_ser\_1.values)

[\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*100%%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*] 1 of 1 completed

### 10.2.2 Виведення графіку досліджуваного ряду

time\_ser.plot(figsize=(8,6), xlabel=xlabel, ylabel=fr"{symbol}")  
plt.savefig('ts1.jpg', bbox\_inches="tight")  
plt.show();



### 10.2.3 Побудова розподілу Леві та розрахунок параметрів для всього ряду

ret\_type = 4  
for\_levy = time\_ser.copy()  
  
if ret\_type == 1:  
 pass  
elif ret\_type == 2:  
 for\_levy = for\_levy.diff()  
elif ret\_type == 3:  
 for\_levy = for\_levy.pct\_change()  
elif ret\_type == 4:  
 for\_levy = for\_levy.pct\_change()  
 for\_levy -= for\_levy.mean()  
 for\_levy /= for\_levy.std()  
elif ret\_type == 5:   
 for\_levy = for\_levy.pct\_change()  
 for\_levy -= for\_levy.mean()  
 for\_levy /= for\_levy.std()  
 for\_levy = for\_levy.abs()  
elif ret\_type == 6:  
 for\_levy -= for\_levy.mean()  
 for\_levy /= for\_levy.std()  
  
for\_levy = for\_levy.dropna().values

### 10.2.4 Підганяємо розподіл Леві та Гаусовий для порівняння

params = levy.fit\_levy(for\_levy)  
mean, std = norm.fit(for\_levy)

### 10.2.5 Отримуємо параметри розподілу Леві у відповідності до однієї із параметризацій, що пропонує пакет

alpha, beta, mu, sigma = params[0].get('1')

### 10.2.6 Будуємо теоретичні та емпіричні розподіли

fig, ax = plt.subplots(1, 2, figsize=(20, 10))  
  
ax[0].hist(for\_levy, bins=50, density=True, alpha=0.6, color='b')  
  
xmin = for\_levy.min()  
xmax = for\_levy.max()  
  
x = np.linspace(xmin, xmax, len(for\_levy))  
pdf = levy.levy(x, alpha, beta, mu, sigma)  
pdf\_norm = norm.pdf(x, mean, std)  
  
fig.suptitle(fr'Теоретичні та емпіричні $\alpha$-стабільні розподіли для {symbol}', fontsize=20)  
  
ax[0].plot(x, pdf, 'k', linewidth=2)  
ax[0].plot(x, pdf\_norm, 'r', linewidth=2)  
ax[0].set\_yscale('log')  
ax[0].set\_xlabel(r'$ x $')  
ax[0].set\_ylabel(r'$ f\_{\alpha}(x), \, \mathrm{ePDF}$')  
  
ax[1].hist(for\_levy, bins=50, density=True, alpha=0.6, color='g')  
ax[1].plot(x, pdf, 'k', linewidth=2)  
ax[1].plot(x, pdf\_norm, 'r', linewidth=2)  
ax[1].set\_xlabel(r'$ x $')  
ax[1].set\_ylabel(r'$ f\_{\alpha}(x), \, \mathrm{ePDF}$')  
  
  
plt.show();

|  |
| --- |
| Рисунок 10.1: Теоретичні та емпіричні альфа-стабільні функції щільності ймовірностей |

### 10.2.7 Виводимо параметри розподілу Леві для заданого індексу

print(fr"Параматери alpha = {alpha:.2f}, beta = {beta:.2f}, mu = {mu:.2f}, sigma = {sigma:.2f}")

Параматери alpha = 1.63, beta = -0.11, mu = -0.01, sigma = 0.53

Для досліджуваного індексу бачимо, що параметр та , що вказує на відхилення розподілу даного індексу від нормального. Тобто, для даного ряду переважаючими є кризові явища, на що вказують важкі хвости розподілу. Із порівняльного аналізу Гауссового та Леві розподілів бачимо, що хвости нормального розподілу значно недооцінюють ймовірність появи кризових явищ чого, наприклад, не скажешь про альфа-стабільний розподіл. Взявши логарифм значень ймовірності по осі ми можемо спостерігати, що, наприклад, недооцінка негативних прибутковостей Гауссовим розподілом, у порівнянні з альфа-стабільним, складає порядків. Для позитивних прибутковостей, що перевищують значення недооцінка Гауссовим розподілом складає порядків. Теоретичне значення альфа-стабільного розподілу достатньо точно враховує важкі хвости емпіричного розподілу, що також виражається високим ексцесом розподілу. Також варто зазначити, що коефіцієнт асиметрії вказує на невеличке зміщення розподілу в ліву сторону, що також демонструє переважання кризових явищ.

### 10.2.8 Дослідження поведінки альфа-стабільного розподілу Леві

x = np.arange(-5, 5, .01)  
beta\_1 = 0  
mu = 0   
sigm = 1   
  
beta\_2 = 1.0  
  
fig, ax = plt.subplots(2, 2, figsize=(20, 10))  
  
ax[0][0].plot(x, levy.levy(x, 0.5, beta\_1, mu, sigm), label = r"$ \alpha=0.5 $")  
ax[0][0].plot(x, levy.levy(x, 0.75, beta\_1, mu, sigm), label = r"$ \alpha=0.75 $")  
ax[0][0].plot(x, levy.levy(x, 1.0, beta\_1, mu, sigm), label = r"$ \alpha=1.0 $")  
ax[0][0].plot(x, levy.levy(x, 1.25, beta\_1, mu, sigm), label = r"$ \alpha=1.25 $")  
ax[0][0].plot(x, levy.levy(x, 1.5, beta\_1, mu, sigm), label = r"$ \alpha=1.5 $")  
  
ax[0][0].set\_title(r"Симетричні $\alpha$-стабільні розподіли, $ \beta = 0 $, $ \mu = 0 $, $ \sigma = 1 $", y=1.03, fontsize=20)  
  
ax[0][0].legend(fontsize=20)  
  
ax[0][0].set\_xlabel(r"$ x $")  
ax[0][0].set\_ylabel(r"$ f\_{\alpha}(x) $")  
  
  
ax[0][1].plot(x, levy.levy(x, 0.5, beta\_2, mu, sigm), label = r"$ \alpha=0.5 $")  
ax[0][1].plot(x, levy.levy(x, 0.75, beta\_2, mu, sigm), label = r"$ \alpha=0.75 $")  
ax[0][1].plot(x, levy.levy(x, 1.0, beta\_2, mu, sigm), label = r"$ \alpha=1.0 $")  
ax[0][1].plot(x, levy.levy(x, 1.25, beta\_2, mu, sigm), label = r"$ \alpha=1.25 $")  
ax[0][1].plot(x, levy.levy(x, 1.5, beta\_2, mu, sigm), label = r"$ \alpha=1.5 $")  
  
ax[0][1].set\_title(r"Зміщенні $\alpha$-стабільні розподіли, $ \beta = 1 $, $ \mu = 0 $, $ \sigma = 1 $", y=1.03, fontsize=20)  
  
ax[0][1].legend(fontsize=20)  
  
ax[0][1].set\_xlabel(r"$ x $")  
ax[0][1].set\_ylabel(r"$ f\_{\alpha}(x) $")  
  
  
ax[1][0].plot(x, levy.levy(x, 0.5, beta\_1, mu, sigm, cdf=True), label = r"$ \alpha=0.5 $")  
ax[1][0].plot(x, levy.levy(x, 0.75, beta\_1, mu, sigm, cdf=True), label = r"$ \alpha=0.75 $")  
ax[1][0].plot(x, levy.levy(x, 1.0, beta\_1, mu, sigm, cdf=True), label = r"$ \alpha=1.0 $")  
ax[1][0].plot(x, levy.levy(x, 1.25, beta\_1, mu, sigm, cdf=True), label = r"$ \alpha=1.25 $")  
ax[1][0].plot(x, levy.levy(x, 1.5, beta\_1, mu, sigm, cdf=True), label = r"$ \alpha=1.5 $")  
  
ax[1][0].set\_title(r"Симетричні $\alpha$-стабільні розподіли, $ \beta = 0 $, $ \mu = 0 $, $ \sigma = 1 $", y=1.03, fontsize=20)  
  
ax[1][0].legend(fontsize=20, loc="lower right")  
  
ax[1][0].set\_xlabel(r"$ x $")  
ax[1][0].set\_ylabel(r"$ F\_{\alpha}(x) $")  
  
ax[1][1].plot(x, levy.levy(x, 0.5, beta\_2, mu, sigm, cdf=True), label = r"$ \alpha=0.5 $")  
ax[1][1].plot(x, levy.levy(x, 0.75, beta\_2, mu, sigm, cdf=True), label = r"$ \alpha=0.75 $")  
ax[1][1].plot(x, levy.levy(x, 1.0, beta\_2, mu, sigm, cdf=True), label = r"$ \alpha=1.0 $")  
ax[1][1].plot(x, levy.levy(x, 1.25, beta\_2, mu, sigm, cdf=True), label = r"$ \alpha=1.25 $")  
ax[1][1].plot(x, levy.levy(x, 1.5, beta\_2, mu, sigm, cdf=True), label = r"$ \alpha=1.5 $")  
  
ax[1][1].set\_title(r"Зміщенні $\alpha$-стабільні розподіли, $\beta = 1$, $\mu = 0$, $\sigma = 1$", y=1.03, fontsize=20)  
  
ax[1][1].legend(fontsize=20, loc="lower right")  
  
ax[1][1].set\_xlabel(r"$ x $")  
ax[1][1].set\_ylabel(r"$ F\_{\alpha}(x) $")  
  
fig.tight\_layout()  
plt.show();

|  |
| --- |
| Рисунок 10.2: Залежність функції щільності ймовірностей альфа-стабільного розподілу Леві та кумулятивної функції щільності від різних значень параметрів розподілу |

### 10.2.9 Задання ширини вікна та кроку

У даному блоці оберемо тип ряду для якого і виконуватимуться розрахунки. Перед нами представлено 6 варіантів представлення часового ряду. Виконуватимемо подальші обчислення для стандартизованих прибутковостей, оскільки згідно багатьом роботам було показано, що розподіл прибутковостей тих же самих фінансових активів (фондових індексів, валютних, криптовалют, тощо) виходить за межі нормального Гауссового розподілу. Покажемо це у даному ноутбуці та застосуємо альфа-стабільний розподіл Леві для кращого моделювання складних систем та передчасної ідентифікації кризових явищ.

window = 250 # ширина сковзного вікна  
tstep = 1 # крок  
  
ret\_type = 4 # тип ряду: 1 - вихідний, 2 - детрендований (різниця між теперішнім значенням та попереднім)  
 # 3 - стандартизовані прибутковості, 4 - стандартизований ряд, 5 - абсолютні значення (волатильність)  
 # 6 - стандартизований вихідний ряд  
   
length = len(time\_ser)

### 10.2.10 Розраховуємо параметри Леві, використовуючи алгоритм сковзного вікна

alpha = []  
beta = []  
mu = []  
sigma = []

for i in tqdm(range(0,length-window,tstep)):  
 fragm = time\_ser.iloc[i:i+window].copy() # відбираємо фрагмент та в подальшому відбираємо потрібний тип ряду  
 if ret\_type == 1:  
 pass  
 elif ret\_type == 2:  
 fragm = fragm.diff()  
 elif ret\_type == 3:  
 fragm = fragm.pct\_change()  
 elif ret\_type == 4:  
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
 elif ret\_type == 5:   
 fragm = fragm.pct\_change()  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
 fragm = fragm.abs()  
 elif ret\_type == 6:  
 fragm -= fragm.mean()  
 fragm /= fragm.std()  
   
 fragm = fragm.dropna().values   
   
 params = levy.fit\_levy(fragm)  
   
 a, b, m, s = params[0].get('0')  
   
 alpha.append(a)  
 beta.append(b)  
 mu.append(m)  
 sigma.append(s)

100%|██████████| 5995/5995 [24:13<00:00, 4.12it/s]

### 10.2.11 Зберігаємо значення у текстовому документі

np.savetxt(f"alpha\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}.txt", alpha)  
np.savetxt(f"beta\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}.txt", beta)  
np.savetxt(f"mu\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}.txt", mu)  
np.savetxt(f"sigma\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}.txt", sigma)

### 10.2.12 Визначимо функцію для побудови парних графіків

def plot\_pair(x\_values,   
 y1\_values,  
 y2\_values,   
 y1\_label,   
 y2\_label,  
 x\_label,   
 file\_name, clr="magenta"):  
  
 fig, ax = plt.subplots()  
  
 ax2 = ax.twinx()  
  
 ax2.spines.right.set\_position(("axes", 1.03))  
  
 p1, = ax.plot(x\_values,   
 y1\_values,   
 "b-", label=fr"{y1\_label}")  
 p2, = ax2.plot(x\_values,  
 y2\_values,   
 color=clr,   
 label=y2\_label)  
  
 ax.set\_xlabel(x\_label)  
 ax.set\_ylabel(f"{y1\_label}")  
  
 ax.yaxis.label.set\_color(p1.get\_color())  
 ax2.yaxis.label.set\_color(p2.get\_color())  
  
 tkw = dict(size=2, width=1.5)  
  
 ax.tick\_params(axis='x', \*\*tkw)  
 ax.tick\_params(axis='y', colors=p1.get\_color(), \*\*tkw)  
 ax2.tick\_params(axis='y', colors=p2.get\_color(), \*\*tkw)  
  
  
 ax2.legend(handles=[p1, p2])  
  
 plt.savefig(file\_name + ".jpg")  
   
 plt.show();

### 10.2.13 Виводимо динаміку показника стабільності

measure\_label = r'$\alpha$'  
file\_name = f"alpha\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 alpha,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name)

|  |
| --- |
| Рисунок 10.3: Динаміка фондового індексу BSESN та показника стабільності |

Параметр (індекс стабільності хвостів розподілу) починає спадати у (перед)кризовий період, що робить його індикатором(-передвісником) кризових явищ. Під час криз у розподілі прибутковостей зростає ексцес, а самі хвости стають важчими, на що даний показник реагує передчасно.

Наступним розглянемо параметр асиметрії :

### 10.2.14 Виводимо динаміку індексу асиметрії

measure\_label = r'$\beta$'  
file\_name = f"beta\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 beta,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name)

|  |
| --- |
| Рисунок 10.4: Динаміка фондового індексу BSESN та показника асиметрії |

Динаміка даної міри виглядає набагато хаотичніше у порівнянні з індексом стабільності. Для представлених результатів можна зробити наступний висновок: у передкризовий період даний показник має зростати, вказуючи на значну правосторонню асиметрію розподілу прибутковостей (переважання позитивних прибутковостей). Для кризового періоду цей показник має спадати, вказуючи на домінацію негативних прибутковостей (лівостороння асиметрія розподілу). Даний показник важко розглядати у якості надійного індикатора, оскільки його коливання представляються значними навіть при незначних падіннях представленого фондового індексу.

Наступним розглянемо індекс розташування (зміщення) альфа-стабільного розподілу:

### 10.2.15 Виводимо динаміку параметру зміщення

measure\_label = r'$\mu$'  
file\_name = f"mu\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 mu,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name)

|  |
| --- |
| Рисунок 10.5: Динаміка фондового індексу BSESN та показника зміщення |

Показник розташування потроху спадає у кризовий період, демонструючи зміщення розподілу в сторону негативних прибутковостей. Тим не менш, цей розподіл представляєть настільки ж хаотичним як і показник асиметрії .

Останні рисунки демонструють динаміку показника масштабування розподілу :

### 10.2.16 Виводимо динаміку параметра масштабу

measure\_label = r'$\sigma$'  
file\_name = f"sigma\_idx\_{symbol}\_{window}\_{tstep}\_{ret\_type}"

plot\_pair(time\_ser.index[window:length:tstep],  
 time\_ser.values[window:length:tstep],  
 sigma,   
 ylabel,   
 measure\_label,  
 xlabel,  
 file\_name)

|  |
| --- |
| Рисунок 10.6: Динаміка фондового індексу BSESN та показника масштабу |

Із представлених результатів видно, що даний показник спадає у (перед)кризовий період, вказуючи на зменшення масштабу (форми) альфа-стабільного розподілу Леві.

Отже, з усіх 4-ох показників, показник стабільності є найкращим для ідентифікації кризових явищ та побудови надійних стратегій ризик-менеджменту.

## 10.3 Висновок

Стабільні розподіли — захоплюючий і плідний об’єкт досліджень в теорії ймовірностей; більше того, в даний час вони мають велику цінність при моделюванні складних процесів у фізиці, астрономії, економіці, теорії комунікацій, тощо.

У даній роботі було представлено теоретичні та чисельні обгрунтування в сторону альфа-стабільного розподілу Леві в якості практичної моделі для кращого розуміння та передбачення кризових явищ у складних системах.

Тут ми представили розрахунки як для усього ряду, так і для його підфрагментів, використовуючи алгоритм ковзного вікна. Виходячи з усього ряду прибутковостей, видно, що хвости їх розподілу далеко виходять за межі Гаусового. Найкраще емпіричний розподіл прибутковостей збігається саме з теоретичним альфа-стабільним розподілом Леві.

Використовуючи алгоритм сковзного вікна, ми побачили, що параметри альфа-стабільного розподілу змінюються з часом.

|  |
| --- |
| Література для подальшого вивчення степеневих розподілів та важких хвостів |
| Nassim Taleb’s [*Statistical Consequences of Fat Tails: Real World Preasymptotics, Epistemology, and Applications*](https://arxiv.org/ftp/arxiv/papers/2001/2001.10488.pdf) (вільно доступна за посиланням). |

# Appendix A — Інструкція зі встановлення Anaconda Navigator

1. Відвідайте сторінку Anaconda за наступним [посиланням](https://www.anaconda.com/download/). Перейшовши, ви маєте побачити наступне зображення:

|  |
| --- |
|  |

1. Натискаємо на кнопку **Download**:

|  |
| --- |
|  |

1. Якщо ви набиратимете назву **anaconda** у пошуковому рядку, тоді потрібно буде перейти за наступним посиланням, що має з’явитися найпершим:

|  |
| --- |
|  |

Перейшовши по виділеному посиланню, ви маєте побачити наступну сторінку:

|  |
| --- |
|  |

Так само як і в пункті 2, потрібно натиснути на кнопку **Download**, щоб розпочати встановлення.

1. З’явиться вікно наступного виду, що запропонує зберегти встановлювальний файл там, де це потрібно:

|  |
| --- |
|  |

1. Натискаємо на клавішу **Next**:

|  |
| --- |
|  |

1. Уважно читаємо ліцензійну угоду та натискаємо клавішу **I Agree**, якщо угода вас задовольняє або вам потрібно скласти залік/іспит:

|  |
| --- |
|  |

1. Далі, якщо ви не перебуваєте у команді розробників, і ви єдина людина, хто буде користуватися Anaconda — **Just me** ваш вибір:

|  |
| --- |
|  |

Натискаємо **Next**.

1. На наступному кроці нам пропонується обрати папку, до якої буде встановлено дистрибутив Anaconda. Бажано, щоб шлях до папки не містив кирилиці. Можна залишити шлях за замовчуванням. Ми натиснемо на **Browse**. Далі, обираємо зручний для нас диск та натискаємо **Создать папку**. Таким чином ми створимо в нашому диску папку з назвою anaconda до якої і буде завантажено Anaconda.

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

1. Наступним кроком буде обрання середовища змінних (the environment variables).

* Якщо ви встановлюєте Python вперше, відмітимо **Add Anaconda to my PATH environment variable**. Окрім всього, це дасть вам можливість використовувати Anaconda в командному рядку (або Git bash, cmder, powershell і т.д.).
* Якщо ви вже маєте Python на своєму комп’ютері, тоді прапорець не варто відмічати. Ви матимете запускати Anaconda Navigator або Anaconda Command Prompt (розташований в меню **Пуск**, у розділі Anaconda) коли вам потрібно буде запускати Anaconda (ви завжди матимете можливість додати Anaconda до свого шляху пізніше, якщо не встановите цей прапорець).
* На представленій локальній машині нам не треба відмічати прапорець.

|  |  |  |
| --- | --- | --- |
| |  | | --- | | Рекомендований підхід | |  |

|  |  |
| --- | --- |
| |  | | --- | | Альтернативний | |

1. Натискаємо **Install** і чекаємо завершення:

|  |  |
| --- | --- |
|  |  |

1. Натискаємо **Next** аж до вікна з подякою за встановлення Anaconda і після натискаємо **Finish**:

|  |
| --- |
|  |

1. На наступному кроці потрібно перейти в меню **Пуск** і знайти папку під назвою Anaconda:

|  |
| --- |
|  |

1. Розгорнувши папку можна побачити цікаві для нас іконки:

|  |
| --- |
|  |

На зазначеній вище фотографії можна бачити декілька ярликів Anaconda Navigator, Anaconda Powershell Promt, Anaconda Promt та Jupyter Notebook з різними найменуваннями **anaconda** та **anaconda3**. Це тому, що на представленій локальній машині попередньо було встановлено Anaconda, але на іншому локальному диску. Якщо ви встановлюєте Anaconda вперше чи перевстановлюєте, тоді ви матимете у 2 рази менше ярликів.

1. Натиснувши на Anaconda Navigator, потрапляємо до середовища, що пропонує різноманітні інструменти для аналізу даних. Для подальшої роботи нам знадобиться лише Jupyter Notebook:

|  |
| --- |
|  |

Натискаємо на **Launch** та потрапляємо до середовища Jupyter (кореневої папки до якої було встановлено anaconda):

|  |
| --- |
|  |

У верхньому лівому кутку буде знаходитись значок **New**. Спочатку натискаємо на нього і потім на **Python 3**. Має створитися відповідний Notebook у якому можна писати код:



1. Окрім цього, в меню пуск можна натиснути на значок Jupyter Notebook та одразу перейти до роботи:

|  |
| --- |
|  |

1. Натиснувши на Anaconda Powershell Promt або Anaconda Promt можна перейти до командного рядка представленого дистрибутивом Anaconda. З його допомогою можна докачати необхідні модулі або запустити необхідний інструмент, що представляє Anaconda. Запустимо Jupyter Notebook за допомогою команди **jupyter notebook**:

# Appendix B — Вступ до мови програмування Python

## B.1 Коментарі коду

Коментар — це примітка, зроблена програмістом у вихідному коді програми. Його мета — прояснити вихідний код і полегшити людям відстеження того, що відбувається. Все, що міститься в коментарі, зазвичай ігнорується при фактичному запуску коду, що робить коментарі корисними для включення пояснень і міркувань, а також для видалення певних рядків коду, в яких ви можете бути не впевнені. Коментарі в Python створюються за допомогою символу решітуи (# вставити текст тут). Включення # у рядок коду коментує все, що слідує за ним.

# print("Привіт усім")  
# Це коментар  
# Ці рядки коду не змінять жодних значень  
# все, що слідує за першим #, не виконується як код

Ви можете побачити текст, укладений у потрійні лапки (""" вставте текст тут """). Такий синтаксис представлятиме багаторядкове коментування, але це не зовсім точно. Це особливий тип string (тип даних, який ми розглянемо), який називаєтьсяdocstring, який використовується для пояснення призначення функції.

""" This is a special string """

' This is a special string '

Переконайтеся, що Ви прочитали коментарі в кожній комірці коду (якщо вони там є). Вони нададуть більш детальні пояснення того, що відбувається в режимі реального часу, коли ви переглядаєте кожен рядок коду.

## B.2 Змінна

Змінні надають імена для значень. Якщо ви хочете зберегти значення для подальшого або повторного використання, ви присвоюєте значенню ім’я, зберігаючи вміст у змінній. Змінні в програмуванні працюють принципово аналогічно змінним в алгебрі, але в Python вони можуть приймати різні типи даних.

Основними типами змінних, які ми розглянемо в цьому розділі, є цілі числа, числа з плаваючою комою, логічні значення та рядки.

Ціле число у програмуванні - це те саме, що і в математиці, кругле число без значень після десяткової коми. Ми використовуємо вбудовану функцію print() тут для відображення значень наших змінних, а також їх типів!

my\_integer = 50  
print(my\_integer, type(my\_integer))

50 <class 'int'>

Змінні, незалежно від типу, призначаються за допомогою одного знака рівності (=). Змінні чутливі до регістру, тому будь-які зміни в зміні заголовних літер імені змінної будуть посилатися повністю на іншу змінну.

one = 1  
print(one)

1

Число з плаваючою комою або float - це вигадлива назва дійсного числа (знову ж таки, як у математиці). Щоб визначити float, нам потрібно або включити десяткову крапку, або вказати, що значення є float.

my\_float = 1.0  
print(my\_float, type(my\_float))  
my\_float = float(1)  
print(my\_float, type(my\_float))

1.0 <class 'float'>  
1.0 <class 'float'>

Змінна типу float не округлятиме число, яке ви в ній зберігаєте, тоді як змінна типу integer округлятиме. Це робить floats більш придатними для математичних обчислень, де потрібно більше, ніж просто цілі числа.

Зверніть увагу, що оскільки ми використовували функцію float(), щоб змусити число рахуватися float, ми можемо використовувати функцію int(), щоб змусити число представлятися в типі int.

my\_int = int(3.14159)  
print(my\_int, type(my\_int))

3 <class 'int'>

Функція int() також усіче будь-які цифри, які число може містити після десяткової коми!

Рядки дозволяють включати текст як змінну для роботи. Вони визначаються з використанням або одинарних лапок (’’), або подвійних лапок (““).

my\_string = 'This is a string with single quotes'  
print(my\_string)  
my\_string = "This is a string with double quotes"  
print(my\_string)

This is a string with single quotes  
This is a string with double quotes

Обидва варіанти дозволені, так що ми можемо включити апострофи або лапки в рядок, якщо ми того побажаємо.

my\_string = '"Jabberwocky", by Lewis Carroll'  
print(my\_string)  
my\_string = "'Twas brillig, and the slithy toves / Did gyre and gimble in the wabe;"  
print(my\_string)

"Jabberwocky", by Lewis Carroll  
'Twas brillig, and the slithy toves / Did gyre and gimble in the wabe;

Логічні значення, або bools, - це двійкові типи змінних. bool може приймати лише одне з двох значень, це True або False. У цій ідеї істинних значень є набагато більше, коли мова заходить про програмування, про що ми розповімо пізніше в розділі [Логічні оператори](#id-section5) цього зошита.

my\_bool = True  
print(my\_bool, type(my\_bool))

True <class 'bool'>

Існує ще багато типів даних, які ви можете призначити змінними в Python, але це основні з них! Ми розглянемо ще трохи пізніше, коли будемо просуватися по цьому блокноту.

## B.3 Базова математика

Python має ряд вбудованих математичних функцій. Їх можна ще більше розширити, імпортуючи пакет **math** або включивши будь-яку кількість інших обчислювальних пакетів.

Підтримуються всі основні арифметичні операції: +, -, /, і \*. Ви можете створювати експоненти за допомогою \*\*, а модульна арифметика вводиться за допомогою оператора mod, %.

print('Addition: ', 2 + 2)  
print('Subtraction: ', 7 - 4)  
print('Multiplication: ', 2 \* 5)  
print('Division: ', 10 / 2)  
print('Exponentiation: ', 3\*\*2)

Addition: 4  
Subtraction: 3  
Multiplication: 10  
Division: 5.0  
Exponentiation: 9

Якщо ви не знайомі з оператором mod, він працює як функція залишку. Якщо ми введемо , він поверне залишок після ділення на .

print('Частка: ', 15 % 4)

Частка: 3

Математичні функції також працюють зі змінними!

first\_integer = 4  
second\_integer = 5  
print(first\_integer \* second\_integer)

20

Переконайтеся, що ваші змінні є плаваючими, якщо ви хочете, щоб у вашій відповіді були десяткові крапки. Якщо ви виконуєте математику виключно з цілими числами, Ви отримуєте ціле число. Включення будь-якого значення з плаваючою точкою в обчислення зробить результат плаваючим.

first\_integer = 11  
second\_integer = 3  
print(first\_integer / second\_integer)

3.6666666666666665

first\_number = 11.0  
second\_number = 3.0  
print(first\_number / second\_number)

3.6666666666666665

Python має кілька вбудованих математичних функцій. Найбільш помітними з них є:

* abs()
* round()
* max()
* min()
* sum()

Усі ці функції діють так, як ви очікували, враховуючи їх назви. Виклик abs() для числа поверне його абсолютне значення. Функція round() округлить число до вказаної кількості десяткових знаків (значення за замовчуванням дорівнює ). Виклик max() або min() для набору чисел поверне, відповідно, максимальне або мінімальне значення в наборі. Виклик sum() для набору чисел призведе до їх підсумовування. Якщо ви не знайомі з тим, як працюють колекції значень у Python, не хвилюйтеся! Ми детально розглянемо набір в наступному розділі.

Додаткові математичні функції можуть бути додані разом з пакетом math.

import math

Математична бібліотека додає довгий список нових математичних функцій до Python. Не соромтеся ознайомитися з [документацією](https://docs.python.org/3/library/math.html) для отримання повного списку та деталей. У ньому полягають деякі математичні константи

print('Pi: ', math.pi)  
print("Euler's Constant: ", math.e)

Pi: 3.141592653589793  
Euler's Constant: 2.718281828459045

А також деякі часто використовувані математичні функції

print('Cosine of pi: ', math.cos(math.pi))

Cosine of pi: -1.0

## B.4 Колекції

### B.4.1 Списки (lists)

Список у Python - це впорядкована колекція об’єктів, яка може містити будь-який тип даних. Ми визначаємо список, використовуючи квадратні дужки ([]).

my\_list = [1, 2, 3]  
print(my\_list)

[1, 2, 3]

Ми також можемо отримати доступ до списку та проіндексувати його за допомогою дужок. Щоб вибрати окремий елемент, просто введіть назву списку, а потім індекс елемента, який ви шукаєте, у фігурних дужках.

print(my\_list[0])  
print(my\_list[2])

1  
3

Індексація в Python починається з $ 0$. Якщо у вас є список довжиною , перший елемент списку знаходиться з індексом , другий елемент з індексом , і так далі, і тому подібне. Останній елемент списку матиме індекс . Будьте обережні! Спроба отримати доступ до неіснуючого індексу призведе до помилки.

print('The first, second, and third list elements: ', my\_list[0], my\_list[1], my\_list[2])  
print('Accessing outside the list bounds causes an error: ', my\_list[3])

The first, second, and third list elements: 1 2 3

IndexError: list index out of range

Ми можемо побачити кількість елементів у списку, викликавши функцію len().

print(len(my\_list))

3

Ми можемо оновлювати та змінювати список, отримуючи доступ до індексу та призначаючи нове значення.

print(my\_list)  
my\_list[0] = 42  
print(my\_list)

[1, 2, 3]  
[42, 2, 3]

Це принципово відрізняється від того, як обробляються рядки. Список є змінним, що означає, що ви можете змінювати елементи списку без зміни самого списку. Деякі типи даних, такі як рядки, є незмінними, що означає, що ви взагалі не можете їх змінити. Як тільки рядок або інший незмінний тип даних був створений, він не може бути безпосередньо змінений без створення абсолютно нового об’єкта.

my\_string = "Strings never change"  
my\_string[0] = 'Z'

TypeError: 'str' object does not support item assignment

Як ми вже говорили раніше, список може містити будь-який тип даних. Таким чином, списки також можуть містити рядки.

my\_list\_2 = ['one', 'two', 'three']  
print(my\_list\_2)

['one', 'two', 'three']

Списки також можуть містити кілька різних типів даних одночасно!

my\_list\_3 = [True, 'False', 42]

Якщо ви хочете об’єднати два списки, їх можна об’єднати символом +.

my\_list\_4 = my\_list + my\_list\_2 + my\_list\_3  
print(my\_list\_4)

[42, 2, 3, 'one', 'two', 'three', True, 'False', 42]

Окрім доступу до окремих елементів списку, ми можемо отримати доступ до груп елементів за допомогою зрізу.

my\_list = ['friends', 'romans', 'countrymen', 'lend', 'me', 'your', 'ears']

#### B.4.1.1 Зріз (slicing)

Ми використовуємо двокрапку (:) для нарізки списків.

print(my\_list[2:4])

['countrymen', 'lend']

Використовуючи :, ми можемо вибрати групу елементів у списку, починаючи з першого вказаного елемента і закінчуючи (але не включаючи) останнім зазначеним елементом.

Ми також можемо вибрати все після певного значення

print(my\_list[1:])

['romans', 'countrymen', 'lend', 'me', 'your', 'ears']

І все перед конкретним значенням

print(my\_list[:4])

['friends', 'romans', 'countrymen', 'lend']

Використання негативних чисел буде відлічуватися з кінця індексів, а не з початку. Наприклад, індекс -1 вказує на останній елемент списку.

print(my\_list[-1])

ears

Ви також можете додати третій компонент для нарізки. Замість того, щоб просто вказати першу та кінцеву частини вашого зрізу, ви можете вказати розмір кроку, який ви хочете зробити. Таким чином, замість того, щоб брати кожен окремий елемент, ви можете взяти будь-який інший елемент.

print(my\_list[0:7:2])

['friends', 'countrymen', 'me', 'ears']

Тут ми вибрали весь список (оскільки 0:7 дасть елементи від 0 до 6), і ми вибрали розмір кроку 2. Отже, це виведе елемент 0, елемент 2, елемент 4 тощо на вибраний елемент списку. Ми можемо пропустити вказаний початок і кінець нашого фрагмента, вказавши лише крок, якщо хочемо.

print(my\_list[::2])

['friends', 'countrymen', 'me', 'ears']

Списки неявно вибирають початок і кінець списку, якщо не вказано інше.

print(my\_list[:])

['friends', 'romans', 'countrymen', 'lend', 'me', 'your', 'ears']

При негативному розмірі кроку ми можемо навіть перевернути список!

print(my\_list[::-1])

['ears', 'your', 'me', 'lend', 'countrymen', 'romans', 'friends']

Python не має власних матриць. Інші пакети, такі як numpy, додають матриці як окремий тип даних, але в базовому Python найкращим способом створення матриці є використання списку списків.

Ми також можемо використовувати вбудовані функції для створення списків. Зокрема, ми розглянемо range() (тому що ми будемо використовувати його пізніше!). Діапазон може приймати кілька різних вхідних даних і поверне список.

b = 10  
my\_list = range(b)  
print(my\_list)

range(0, 10)

Подібно до наших попередніх методів нарізки списків, ми можемо визначити як початок, так і кінець нашого діапазону. Це поверне список, який включає початок і виключає кінець, точно так само, як зріз.

a = 0  
b = 10  
my\_list = range(a, b)  
print(my\_list)

range(0, 10)

Ми також можемо вказати розмір кроку. Це знову має таку ж поведінку, як і зріз.

a = 0  
b = 10  
step = 2  
my\_list = range(a, b, step)  
print(my\_list)

range(0, 10, 2)

### B.4.2 Кортежі (Tuples)

Кортеж - це тип даних, подібний до списку в тому сенсі, що він може містити різні типи даних. Ключова відмінність тут полягає в тому, що кортеж є незмінним. Ми визначаємо кортеж, розділяючи елементи, які ми хочемо включити комами. Зазвичай кортеж укладають в круглі дужки.

my\_tuple = 'I', 'have', 30, 'cats'  
print(my\_tuple)

('I', 'have', 30, 'cats')

my\_tuple = ('I', 'have', 30, 'cats')  
print(my\_tuple)

('I', 'have', 30, 'cats')

Як згадувалося раніше, кортежі незмінні. Ви не можете змінити будь-яку їх частину, не визначивши новий кортеж.

my\_tuple[3] = 'dogs' # Намагається змінити значення 'cats', що зберігається в кортежі, на 'dogs'

TypeError: 'tuple' object does not support item assignment

Ви можете нарізати кортежі так само, як ви нарізаєте списки!

print(my\_tuple[1:3])

('have', 30)

І об’єднайте їх так, як Ви б це зробили з рядками!

my\_other\_tuple = ('make', 'that', 50)  
print(my\_tuple + my\_other\_tuple)

('I', 'have', 30, 'cats', 'make', 'that', 50)

Ми можемо упакувати значення разом, створивши кортеж (як зазначено вище), або ми можемо розпакувати значення з кортежу, витягуючи їх.

str\_1, str\_2, int\_1 = my\_other\_tuple  
print(str\_1, str\_2, int\_1)

make that 50

Розпакування присвоює кожне значення кортежу по порядку кожній змінній у лівій частині знака рівності. Деякі функції, включаючи спеціальні функції, можуть повертати кортежі, тому ми можемо використовувати це, щоб безпосередньо розпакувати їх і отримати доступ до потрібних нам значень.

### B.4.3 Множини (Sets)

Множини - це набір невпорядкованих, унікальних елементів. Він працює майже точно так, як ви очікували б від звичайного набору математичних задач, і визначається за допомогою фігурних дужок ({}).

things\_i\_like = {'dogs', 7, 'the number 4', 4, 4, 4, 42, 'lizards', 'man I just LOVE the number 4'}  
print(things\_i\_like, type(things\_i\_like))

{'lizards', 'dogs', 'the number 4', 4, 'man I just LOVE the number 4', 7, 42} <class 'set'>

Зверніть увагу, як будь-які додаткові екземпляри одного і того ж елемента видаляються в остаточному наборі. Ми також можемо створити множину зі списку, використовуючи функцію set().

animal\_list = ['cats', 'dogs', 'dogs', 'dogs', 'lizards', 'sponges', 'cows', 'bats', 'sponges']  
animal\_set = set(animal\_list)  
print(animal\_set) # видаляємо всі дублікати зі списку

{'dogs', 'bats', 'lizards', 'cats', 'sponges', 'cows'}

Виклик len() для множини повідомить вам, скільки в ньому елементів.

print(len(animal\_set))

6

Оскільки множина представляє невпорядковану структуру даних, ми не можемо отримати доступ до окремих елементів за допомогою індексу. Однак ми можемо легко перевірити приналежність (щоб побачити, чи міститься щось у наборі) та використовувати об’єднання та перетини множин за допомогою вбудованих функцій set.

'cats' in animal\_set # Тут ми перевіряємо наявність членства, використовуючи ключове слово 'in'.

True

Тут ми перевірили, чи міститься рядок cats у нашому animal\_set, і він повернув True, повідомивши нам, що він насправді знаходиться в нашому наборі.

Ми можемо з’єднати множини, використовуючи типові математичні оператори множин, а саме | для об’єднання та & для перетину. Використання | або & поверне саме те, що ви очікували б, якщо Ви знайомі з множинами в математиці.

print(animal\_set | things\_i\_like) # Ви також можете написати things\_i\_like / animal\_set без будь-якої різниці

{'dogs', 'the number 4', 4, 'bats', 'man I just LOVE the number 4', 7, 42, 'lizards', 'cats', 'sponges', 'cows'}

Сполучення двох наборів за допомогою | об’єднує множини, видаляючи будь-які повторення, щоб зробити кожен елемент набору унікальним.

print(animal\_set & things\_i\_like) # Ви також можете написати things\_i\_like & animal\_set без будь-якої різниці

{'lizards', 'dogs'}

Сполучення двох наборів за допомогою & обчислює перетин обох наборів, повертаючи набір, який містить лише те, що вони мають спільне.

Якщо вам цікаво дізнатися більше про вбудовані функції для наборів, не соромтеся ознайомитися з [документацією](https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset).

### B.4.4 Словники (Dictionaries)

Ще однією важливою структурою даних у Python є словник. Словники визначаються за допомогою комбінації фігурних дужок ({}) і двокрапок (:). Фігурні дужки визначають початок і кінець словника, а двокрапки вказують пари ключ-значення. Словник - це, по суті, набір пар ключ-значення. Ключ будь-якого запису повинен бути незмінним типом даних. Це робить кандидатами як рядки, так і кортежі. Ключі можуть бути як додані, так і видалені.

У наступному прикладі ми маємо словник, що складається з пар ключ-значення, де ключовим є жанр художньої літератури (рядок), а значенням є список книг (list) у цьому жанрі. Оскільки колекція все ще вважається єдиною сутністю, ми можемо використовувати її для збору декількох змінних або значень в одну пару ключ-значення.

my\_dict = {"High Fantasy": ["Wheel of Time", "Lord of the Rings"],  
 "Sci-fi": ["Book of the New Sun", "Neuromancer", "Snow Crash"],  
 "Weird Fiction": ["At the Mountains of Madness", "The House on the Borderland"]}

Після визначення словника ми можемо отримати доступ до будь-якого окремого значення, вказавши його ключ у дужках.

print(my\_dict["Sci-fi"])

['Book of the New Sun', 'Neuromancer', 'Snow Crash']

Ми також можемо змінити значення, пов’язане з даним ключем

my\_dict["Sci-fi"] = "I can't read"  
print(my\_dict)

{'High Fantasy': ['Wheel of Time', 'Lord of the Rings'], 'Sci-fi': "I can't read", 'Weird Fiction': ['At the Mountains of Madness', 'The House on the Borderland']}

Додати нову пару ключ-значення так само просто, як і визначити її.

my\_dict["Historical Fiction"] = ["Pillars of the Earth"]  
print(my\_dict["Historical Fiction"])

['Pillars of the Earth']

print(my\_dict)

{'High Fantasy': ['Wheel of Time', 'Lord of the Rings'], 'Sci-fi': "I can't read", 'Weird Fiction': ['At the Mountains of Madness', 'The House on the Borderland'], 'Historical Fiction': ['Pillars of the Earth']}

## B.5 Рядки (Strings)

Ми вже знаємо, що рядки зазвичай використовуються для тексту. Ми можемо використовувати вбудовані операції для легкого об’єднання, розділення та форматування рядків, залежно від наших потреб.

Символ + вказує на конкатенацію мовою рядків. Це об’єднає два рядки в довший рядок.

first\_string = '"Beware the Jabberwock, my son! /The jaws that bite, the claws that catch! /'  
second\_string = 'Beware the Jubjub bird, and shun /The frumious Bandersnatch!"/'  
third\_string = first\_string + second\_string  
print(third\_string)

"Beware the Jabberwock, my son! /The jaws that bite, the claws that catch! /Beware the Jubjub bird, and shun /The frumious Bandersnatch!"/

Рядки також індексуються приблизно так само, як і списки.

my\_string = 'Supercalifragilisticexpialidocious'  
print('The first letter is: ', my\_string[0]) # Uppercase S  
print('The last letter is: ', my\_string[-1]) # lowercase s  
print('The second to last letter is: ', my\_string[-2]) # lowercase u  
print('The first five characters are: ', my\_string[0:5]) # Remember: slicing doesn't include the final element!  
print('Reverse it!: ', my\_string[::-1])

The first letter is: S  
The last letter is: s  
The second to last letter is: u  
The first five characters are: Super  
Reverse it!: suoicodilaipxecitsiligarfilacrepuS

Вбудовані об’єкти та класи часто мають пов’язані з ними спеціальні функції, які називаються методами. Ми отримуємо доступ до цих методів, використовуючи точку (‘.’). Ми детальніше розглянемо об’єкти та пов’язані з ними методи в іншій лекції!

Використовуючи рядкові методи, ми можемо підраховувати екземпляри символу або групи символів.

print('Count of the letter i in Supercalifragilisticexpialidocious: ', my\_string.count('i'))  
print('Count of "li" in the same word: ', my\_string.count('li'))

Count of the letter i in Supercalifragilisticexpialidocious: 7  
Count of "li" in the same word: 3

Ми також можемо знайти перший екземпляр символу або групи символів у рядку.

print('The first time i appears is at index: ', my\_string.find('i'))

The first time i appears is at index: 8

А також замінити символи в рядку.

print("All i's are now a's: ", my\_string.replace('i', 'a'))

All i's are now a's: Supercalafragalastacexpaaladocaous

print("It's raining cats and dogs".replace('dogs', 'more cats'))

It's raining cats and more cats

Існують також деякі методи, які є унікальними для рядків. Функція upper() перетворює всі символи в рядку в верхній регістр, в той час як lower() перетворює всі символи в рядку в нижній регістр!

my\_string = "I can't hear you"  
print(my\_string.upper())  
my\_string = "I said HELLO"  
print(my\_string.lower())

I CAN'T HEAR YOU  
i said hello

### B.5.1 Форматування рядків

Використовуючи метод format(), ми можемо додавати значення змінних і зазвичай форматувати наші рядки.

my\_string = "{0} {1}".format('Marco', 'Polo')  
print(my\_string)

Marco Polo

my\_string = "{1} {0}".format('Marco', 'Polo')  
print(my\_string)

Polo Marco

Ми використовуємо фігурні дужки ({}) для позначення частин рядка, які будуть заповнені пізніше, і ми використовуємо аргументи функції format() для надання значень для заміни. Цифри у фігурних дужках вказують індекс значення в аргументах format().

Дивіться format() [документація](https://docs.python.org/3/library/string.html#format-examples) для отримання додаткових прикладів.

Якщо вам потрібне швидке та брудне форматування, ви можете замість цього використовувати символ%, який називається оператором форматування рядка.

print('insert %s here' % 'value')

insert value here

Символ % в основному вказує Python на створення заповнювача. Будь-який символ, що слідує за % (у рядку), вказує, який тип матиме значення, введене в заповнювач. Цей символ називається *типом перетворення*. Після закриття рядка нам знадобиться ще один %, за яким слідують значення для вставки. У випадку одного значення ви можете просто помістити його туди. Якщо ви вставляєте більше одного значення, вони повинні бути укладені в кортеж.

print('There are %s cats in my %s' % (13, 'apartment'))

There are 13 cats in my apartment

У цих прикладах %s вказує, що Python повинен перетворити значення в рядки. Існує кілька типів перетворення, які ви можете використовувати, щоб уточнити форматування. Дивіться [форматування рядка](https://docs.python.org/3/library/stdtypes.html#string-formatting) для отримання додаткових прикладів та більш повної інформації про використання.

## B.6 Логічні оператори

### B.6.1 Базова логіка

Логічні оператори мають справу з булевими значеннями, як ми коротко розглянули раніше. Якщо ви пам’ятаєте, bool приймає одне з двох значень: True або False (або або ). Основні логічні твердження, які ми можемо зробити, визначаються за допомогою вбудованих компараторів. Це == (дорівнює), != (не дорівнює), < (Менше), > (Більше), <= (менше або дорівнює) і >= (більше або дорівнює).

print(5 == 5)

True

print(5 > 5)

False

Ці компаратори також працюють у поєднанні зі змінними.

m = 2  
n = 23  
print(m < n)

True

Ми можемо зв’язати ці компаратори разом, щоб створити більш складні логічні оператори, використовуючи логічні оператори or, and і not.

statement\_1 = 10 > 2  
statement\_2 = 4 <= 6  
print("Statement 1 truth value: {0}".format(statement\_1))  
print("Statement 2 truth value: {0}".format(statement\_2))  
print("Statement 1 and Statement 2: {0}".format(statement\_1 and statement\_2))

Statement 1 truth value: True  
Statement 2 truth value: True  
Statement 1 and Statement 2: True

Оператор or виконує логічне обчислення або. Будь-який компонент, об’єднаний за допомогою або, що є True, представлятиме все твердження як True. Оператор and виводить True, лише якщо всі компоненти разом є True. В іншому випадку він видасть False. Твердження not просто інвертує значення істинності будь-якого наступного за ним твердження. Таким чином, твердження True буде оцінено як False, коли перед ним буде поставлено not. Аналогічно, False твердження стане True, коли перед ним буде стояти not.

Припустимо, у нас є два логічні твердження, і . Таблиця істинності для основних логічних операторів виглядає наступним чином:

| P | Q | not P | P and Q | P or Q |
| --- | --- | --- | --- | --- |
| True | True | False | True | True |
| False | True | True | False | True |
| True | False | False | False | True |
| False | False | True | False | False |

Ми можемо зв’язати кілька логічних операторів разом, використовуючи логічні оператори.

print(((2 < 3) and (3 > 0)) or ((5 > 6) and not (4 < 2)))

True

Логічні твердження можуть бути настільки простими або складними, наскільки нам подобається, залежно від того, що нам потрібно висловити. Оцінюючи наведене вище логічне твердження крок за кроком, ми бачимо, що ми оцінюємо (True and True) or (False and not False). Дана конструкція набуваж вигляду True or (False and True). Згодом стає True or False, і в кінцевому рахунку оцінюється як True.

#### B.6.1.1 Істинність

Типи даних у Python мають цікаву характеристику, яка називається істинністю. Це означає, що більшість вбудованих типів будуть оцінюватися як True або False, коли потрібне логічне значення (наприклад, за допомогою оператора if). Як правило, контейнери, такі як рядки, кортежі, словники, списки та множини, повертають True, якщо вони взагалі що-небудь містять, і False, якщо вони нічого не містять.

# Cхоже до того, як працюють float() та int(), book () змушує значення вважатися логічним!  
print(bool(''))

False

print(bool('I have character!'))

True

print(bool([]))

False

print(bool([1, 2, 3]))

True

І так далі, для інших колекцій та контейнерів. None також оцінюється як False. Число 1 еквівалентно True, а число 0 також еквівалентно False в логічному контексті.

### B.6.2 If-оператори

Ми можемо створювати сегменти коду, які виконуються тільки при виконанні набору умов. Ми використовуємо оператори if у поєднанні з логічними операторами для створення розгалужень у нашому коді.

Блок if вводиться, коли умова вважається True. Якщо умова оцінюється як False, блок if буде просто пропущений, якщо тільки до нього не додається блок else. Умови створюються за допомогою логічних операторів або за допомогою істинності значень у Python. Оператор if визначається двокрапкою і блоком тексту з відступом.

if "Condition":  
 print(True)  
else:  
 print(False)

True

i = 4  
if i == 5:  
 print('The variable i has a value of 5')

Оскільки в цьому прикладі i = 4 і оператор if шукає лише те, чи i = 5, оператор print ніколи не буде виконаний. Ми можемо додати оператор else, щоб створити блок коду на випадок надзвичайних ситуацій на випадок, якщо умова в операторі if не буде оцінена як True.

i = 5  
if i == 5:  
 print("Усі рядки в цьому блоці з відступом є частиною цього блоку")  
 print('Змінна i має значення 5')  
else:  
 print("Усі рядки в цьому блоці з відступом є частиною цього блоку")  
 print('Змінна i не дорівнює 5')

Усі рядки в цьому блоці з відступом є частиною цього блоку  
Змінна i має значення 5

Ми можемо реалізувати інші гілки від того самого оператора if, використовуючи elif, скорочення від else if. Ми можемо включати стільки elifсів, скільки захочемо, поки не вичерпаємо всі логічні гілки умови.

i = 1  
if i == 1:  
 print('Змінна i має значення 1')  
elif i == 2:  
 print('Змінна і має значення 2')  
elif i == 3:  
 print('Змінна і має значення 3')  
else:  
 print("Мене не хвилює змінна і")

Мене не хвилює змінна і

Ви також можете вкласти оператори if в інші оператори if, щоб перевірити наявність додаткових умов.

i = 10  
if i % 2 == 0:  
 if i % 3 == 0:  
 print('і ділиться як на 2, так і на 3!')  
 elif i % 5 == 0:  
 print('і ділиться як на 2, так і на 5!')  
 else:  
 print('i ділиться на 2, але не на 3 або 5!')  
else:  
 print('Я припускаю, що i - непарне число.')

і ділиться як на 2, так і на 5!

Пам’ятайте, що ми можемо згрупувати кілька умов разом, використовуючи логічні оператори!

i = 11  
j = 12  
if i < 10 and j > 11:  
 print('{0} менше 10 і {1} більше 11!'.format(i, j))

Ви можете використовувати логічні компаратори для порівняння рядків!

my\_string = "Farthago delenda est"  
if my\_string == "Carthago delenda est":  
 print('And so it was! For the glory of Rome!')  
else:  
 print('War elephants are TERRIFYING. I am staying home.')

War elephants are TERRIFYING. I am staying home.

Як і у випадку з іншими типами даних, == перевірить, чи дві речі з обох сторін мають однакове значення.

Деякі вбудовані функції повертають логічне значення, тому їх можна використовувати як умови в операторі if. Користувацькі функції також можуть бути сконструйовані таким чином, щоб вони повертали логічне значення. Це буде розглянуто пізніше в розділі визначення функції!

Ключове слово in зазвичай використовується для перевірки приналежності значення до іншого значення. Ми можемо перевірити приналежність у контексті оператора if і використовувати його для виведення значення істини.

if 'a' in my\_string or 'e' in my\_string:  
 print('Those are my favorite vowels!')

Those are my favorite vowels!

Тут ми використовуємо in, щоб перевірити, чи містить змінна my\_string містить якісь конкретні літери. Пізніше ми будемо використовувати in для перебору списків!

## B.7 Циклічні структури

Циклічні структури є однією з найважливіших частин програмування. Цикл for і цикл while надають спосіб багаторазового запуску блоку коду повторно. Цикл while буде повторюватися, поки не буде виконана певна умова. Якщо в будь-який момент після ітерації ця умова більше не виконується, цикл завершується. Цикл for буде виконувати ітерацію по послідовності значень і завершиться, коли послідовність закінчиться. Натомість ви можете включити умови в цикл for, щоб вирішити, чи повинен він закінчуватися достроково, чи ви можете просто дозволити йому піти своїм шляхом.

i = 10  
while i > 0:  
 i -= 1  
 print('Я зациклений! {0} значень до завершення!'.format(i))

Я зациклений! 9 значень до завершення!  
Я зациклений! 8 значень до завершення!  
Я зациклений! 7 значень до завершення!  
Я зациклений! 6 значень до завершення!  
Я зациклений! 5 значень до завершення!  
Я зациклений! 4 значень до завершення!  
Я зациклений! 3 значень до завершення!  
Я зациклений! 2 значень до завершення!  
Я зациклений! 1 значень до завершення!  
Я зациклений! 0 значень до завершення!

За допомогою циклів while нам потрібно переконатися, що щось насправді змінюється від ітерації до ітерації, щоб цикл фактично закінчувався. У цьому випадку ми використовуємо скорочення i -= 1 (скорочення від i = i - 1), так що значення i стає меншим з кожною ітерацією. Врешті-решт i буде зменшено до 0, що призведе до виконання умови False та виходу з циклу.

Цикл for повторюється задану кількість разів, що визначається при вказівці запису в цикл. У цьому випадку ми повторюємо список, повернутий з range(). Цикл for вибирає значення зі списку по порядку і тимчасово присвоює йому значення i, щоб із цим значенням можна було виконувати операції.

for i in range(5):  
 print('Я зациклений! Я вже на {0}-ій ітерації!'.format(i + 1))

Я зациклений! Я вже на 1-ій ітерації!  
Я зациклений! Я вже на 2-ій ітерації!  
Я зациклений! Я вже на 3-ій ітерації!  
Я зациклений! Я вже на 4-ій ітерації!  
Я зациклений! Я вже на 5-ій ітерації!

Зверніть увагу, що в цьому циклі for ми використовуємо ключове слово in. Використання ключового слова in не обмежується перевіркою приналежності, як у прикладі if-конструкцій. Ви можете оброблювати будь-яку колекцію за допомогою циклу for, використовуючи ключове слово in.

У цьому наступному прикладі ми переглянемо множину, оскільки хочемо перевірити наявність вмісту та додати до нового набору.

my\_list = {'cats', 'dogs', 'lizards', 'cows', 'bats', 'sponges', 'humans'}  
mammal\_list = {'cats', 'dogs', 'cows', 'bats', 'humans'} # перераховані всі ссавці в у світі  
my\_new\_list = set()  
for animal in my\_list:  
 if animal in mammal\_list:  
 # додаємо будь-яку тварину, що знаходиться і в my\_list, і в mammal\_list  
 my\_new\_list.add(animal)  
  
print(my\_new\_list)

{'dogs', 'bats', 'cats', 'humans', 'cows'}

Є два твердження, які дуже корисні при роботі як з циклами for, так і з циклами while. Це break і continue. Якщо break трапляється в будь-який момент під час виконання циклу, цикл негайно завершується.

i = 10  
while True:  
 if i == 14:  
 break  
 i += 1  
 print(i)

11  
12  
13  
14

for i in range(5):  
 if i == 2:  
 break  
 print(i)

0  
1

Оператор continue вкаже циклу негайно завершити цю ітерацію і перейти до наступної ітерації циклу.

i = 0  
while i < 5:  
 i += 1  
 if i == 3:  
 continue  
 print(i)

1  
2  
4  
5

Цей цикл пропускає друк числа через інструкцію continue, яка виконується, коли ми вводимо оператор if. Код ніколи не бачить команди для друку числа , оскільки він уже перейшов до наступної ітерації.

Змінна, яку ми використовуємо для ітерації циклу, збереже своє значення при завершенні циклу. Аналогічно, будь-які змінні, визначені в контексті циклу, продовжуватимуть існувати поза ним.

for i in range(5):  
 loop\_string = 'Я виходжу за межі циклу!'  
 print('Я вічний! Я {0} і я існую скрізь!'.format(i))  
  
print('Моє значення {0}'.format(i))  
print(loop\_string)

Я вічний! Я 0 і я існую скрізь!  
Я вічний! Я 1 і я існую скрізь!  
Я вічний! Я 2 і я існую скрізь!  
Я вічний! Я 3 і я існую скрізь!  
Я вічний! Я 4 і я існую скрізь!  
Моє значення 4  
Я виходжу за межі циклу!

Ми також можемо виконувати ітерації по словнику!

my\_dict = {'firstname' : 'Inigo', 'lastname' : 'Montoya', 'nemesis' : 'Rugen'}

for key in my\_dict:  
 print(key)

firstname  
lastname  
nemesis

Якщо ми просто перебираємо словник, не роблячи нічого іншого, ми отримуємо лише ключі. Ми можемо або використовувати ключі для отримання значень, як у прикладі:

for key in my\_dict:  
 print(my\_dict[key])

Inigo  
Montoya  
Rugen

Або ми можемо використовувати функцію items(), щоб отримати і ключ, і значення одночасно

for key, value in my\_dict.items():  
 print(key, ':', value)

firstname : Inigo  
lastname : Montoya  
nemesis : Rugen

Функція items створює кортеж з кожної пари ключ-значення, а цикл for розпаковує цей кортеж в ключ, значення при кожному окремому виконанні циклу!

## B.8 Функції

Функція-це багаторазовий блок коду, який ви можете викликати повторно для виконання обчислень, виведення даних або дійсно робити все, що завгодно. Це один з ключових аспектів використання мови програмування. Щоб додати до вбудованих функцій у Python, ви можете визначити свої власні!

def hello\_world():  
 """ Виводить Hello, world! """  
 print('Hello, world!')  
  
hello\_world()

Hello, world!

for i in range(5):  
 hello\_world()

Hello, world!  
Hello, world!  
Hello, world!  
Hello, world!  
Hello, world!

Функції визначаються за допомогою def, імені функції, списку параметрів та двокрапки. Все, що вказано з відступом нижче двокрапки, буде включено у визначення функції.

Ми можемо змусити наші функції робити все, що ви можете зробити зі звичайним блоком коду. Наприклад, наша функція hello\_world() виводить рядок при кожному його виклику. Якщо ми хочемо зберегти значення, обчислене функцією, ми можемо визначити функцію так, щоб вона return потрібне нам значення. Це дуже важлива особливість функцій, оскільки будь-яка змінна, визначена виключно всередині функції, не буде існувати поза нею.

def see\_the\_scope():  
 return "Я тут застряг!"  
  
print(see\_the\_scope())

Я тут застряг!

a = see\_the\_scope()  
print(a)

Я тут застряг!

**Область** змінної - це частина блоку коду, де ця змінна прив’язана до певного значення. Функції в Python мають закриту область дії, що робить можливим прямий доступ до змінних лише всередині цих областей. Якщо ми передамо ці значення оператору return, ми можемо отримати їх із функції.

def free\_the\_scope():  
 in\_function\_string = "Anything you can do I can do better!"  
 return in\_function\_string  
my\_string = free\_the\_scope()  
print(my\_string)

Anything you can do I can do better!

Так само, як ми можемо отримувати значення з функції, ми також можемо розміщувати значення у функції. Ми робимо це, визначаючи нашу функцію з параметрами.

def multiply\_by\_five(x):  
 """ Множимо вхідне значення на 5 """  
 return x \* 5  
  
n = 4  
print(n)  
print(multiply\_by\_five(n))

4  
20

У цьому прикладі у нас був лише один параметр для нашої функції, x. Ми можемо легко додати додаткові параметри, розділивши всі комою.

def calculate\_area(length, width):  
 """ Визначаємо площу прямокутника """  
 return length \* width

l = 5  
w = 10  
print('Area: ', calculate\_area(l, w))  
print('Length: ', l)  
print('Width: ', w)

Area: 50  
Length: 5  
Width: 10

def calculate\_volume(length, width, depth):  
 """ Визначаємо об'єм прямокутної призми """  
 return length \* width \* depth

Якщо ми хочемо, ми можемо визначити функцію так, щоб вона приймала довільну кількість параметрів. Ми повідомляємо Python, що хочемо цього, використовуючи зірочку (\*).

def sum\_values(\*args):  
 sum\_val = 0  
 for i in args:  
 sum\_val += i  
 return sum\_val

print(sum\_values(1, 2, 3))  
print(sum\_values(10, 20, 30, 40, 50))  
print(sum\_values(4, 2, 5, 1, 10, 249, 25, 24, 13, 6, 4))

6  
150  
343

Ви використовуєте \*args як параметр для вашої функції - це коли ви не знаєте, скільки значень можна передати в неї, як у випадку з нашою функцією sum. Зірочка в даному випадку - це синтаксис, який повідомляє Python, що ви збираєтеся передати довільну кількість параметрів у свою функцію. Ці параметри зберігаються у вигляді кортежу.

def test\_args(\*args):  
 print(type(args))  
  
test\_args(1, 2, 3, 4, 5, 6)

<class 'tuple'>

Наші функції можуть повертати будь-який тип даних. Це дозволяє нам легко створювати функції, які перевіряють умови, які ми можемо захотіти відстежувати.

Тут ми визначаємо функцію, яка повертає логічне значення. Ми можемо легко використовувати це в поєднанні з операторами if та іншими ситуаціями, які потребують логічного значення.

def has\_a\_vowel(word):  
 """  
 Перевіряємо, чи містить слово голосну  
  
 """  
 vowel\_list = ['a', 'e', 'i', 'o', 'u']  
  
 for vowel in vowel\_list:  
 if vowel in word:  
 return True  
  
 return False

my\_word = 'catnapping'  
if has\_a\_vowel(my\_word):  
 print('Містить.')  
else:  
 print('Не містить.')

Містить.

def point\_maker(x, y):  
 """ Групує значення x і y в точку, технічно кортеж """  
 return x, y

Ця наведена вище функція повертає впорядковану пару вхідних параметрів, збережених як кортеж.

a = point\_maker(0, 10)  
b = point\_maker(5, 3)  
def calculate\_slope(point\_a, point\_b):  
 """ Обчислює лінійний нахил між двома точками"""  
 return (point\_b[1] - point\_a[1])/(point\_b[0] - point\_a[0])  
print("Кут нахилу між a і b {0}".format(calculate\_slope(a, b)))

Кут нахилу між a і b -1.4

## B.9 Подальші кроки

Якщо ви хочете глибше заглибитися в матеріал, тоді зверніться до [документації по Python](https://docs.python.org/3/index.html).

# Appendix C — Основи Jupyter Notebook

Jupyter Notebook — це веб додаток для інтерактивних обчислень. Для його запуску необхідно в консолі перейти в потрібну папку і виконати команду jupyter notebook. Після цього відкриється веб сторінка, де будуть відображатися файли тієї директорії, звідки була запущена команда. Файли Юпітера, які також називаються зошитами, мають розширення **ipynb**. При натисканні на такий файл він відкривається в інтерактивному режимі.

Кожен окремий файл представляє веб сторінку, яка складається з комірок. Кожна комірка може бути двох видів: 1. Markdown або 2. Code.

## C.1 Комірка Markdown

Як можна здогадатися з назви в Markdown осередках можна створювати текст в [markdown форматі](http://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Working%20With%20Markdown%20Cells.html). Підтримуються різні способи форматування, які можна подивитися за посиланням. Текст, який ви зараз читаєте, також знаходиться в markdown клітинці.

Крім форматування тексту також підтримується можливість створення математичних формул за допомогою LaTex. Формулу можна вбудувати в текст (наприклад, ) або створити в окремому рядку:

Для редагування тексту в markdown осередку необхідно два рази клікнути по ній.

## C.2 Комірка Code

Наступна комірка є Сode осередком і в ній можна писати код і виконувати його. Для виконання коду необхідно натиснути **Ctrl + Enter**(виконати і залишитися в поточній комірці) або **Shift + Enter** (виконати і перейти в наступну комірку)

import numpy as np # імпортуємо бібліотеку

Якщо останній рядок коду повертає яке-небудь значення, то воно відображається відразу після комірки

np.random.rand(10) # генеруємо випадкові значення

array([0.68528999, 0.70609617, 0.82579621, 0.38756313, 0.87001097,  
 0.91044051, 0.31743928, 0.27789269, 0.79610991, 0.58255746])

## C.3 Автодоповнення та робота з документацією

Для автодоповнення можна використовувати клавішу <TAB > після точки або всередині дужки при виклику функції. При цьому вийде список доступних варіантів, які можна вибрати, щоб автоматично доповнити код. Можете спробувати автодоповнення поставивши курсор після np.random.<TAB>.

В Jupyter є кілька способів викликати документацію. Перший спосіб це використовувати поєднання клавіш **Shift + Tab**. Другий спосіб поставити знак ? після необхідного модуля

np?

## C.4 Magic команди

Jupyter підтримує набір так званих “чарівних” (magic) команд. Це різні корисні команди, які не є частиною Python. Всі ці команди починаються з %.

Можна безпосередньо завантажити вміст зовнішнього файлу в комірку за допомогою команди %load

# %load code/magic\_example.py  
def square(x): # ініціалізуємо функцію знаходження квадрату вхідного значення  
 """  
 Squares given number  
 """  
 return x \*\* 2 # повертаємо значення  
  
  
print(square(42)) # виводимо результат

1764

З корисних команд також можна відзначити команду %timeit, яка виконує код багато разів і виводить середній час виконання коду

%timeit L = [n \*\* 2 for n in range(1000)]

377 µs ± 21.4 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)

Список усіх magic команд можна подивитися окремою командою %lsmagic.

## C.5 Робота з графікою

У Python є багато бібліотек для візуалізації даних. Багато з них інтегруються з Jupyter і відображають графіки

import matplotlib.pyplot as plt # імпортуємо бібліотеку  
  
# вбудовуємо вивдені рисунки в юпітеровський ноутбук  
%matplotlib inline  
  
plt.plot([1, 4], [1, 4]) # виводимо лінію по двум точкам



або декілька графіків

all\_data = [np.random.normal(0, std, size=100) for std in range(1, 4)] # генеруємо список значень із нормального розподілу  
labels = ['x1', 'x2', 'x3'] # ініціалізуємо список міток  
  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(9, 4)) # ініціалізуємо об'єкт матриці рисунків  
  
# прямокутна коробчаста діаграма  
bplot1 = axes[0].boxplot(all\_data,  
 vert=True, # вертикальне вирівнювання  
 patch\_artist=True, # заповнити кольором  
 labels=labels) # використовується для позначення підписів по осі x  
axes[0].set\_title('Прямокутна діаграма') # встановлюємо титулку для першого рисунку  
  
# побудова коробчастої діаграми з виїмкою  
bplot2 = axes[1].boxplot(all\_data,  
 notch=True,  
 vert=True, # вертикальне вирівнювання  
 patch\_artist=True, # заповнити кольором  
 labels=labels) # використовується для позначення підписів по осі x  
axes[1].set\_title('Прямокутна діаграма з виїмкою') # встановлюємо титулку для другого рисунку  
  
# заповнити кольорами  
colors = ['pink', 'lightblue', 'lightgreen']  
for bplot in (bplot1, bplot2):  
 for patch, color in zip(bplot['boxes'], colors):  
 patch.set\_facecolor(color)  
  
# додати сітку з горизонтальних ліній  
for ax in axes:  
 ax.yaxis.grid(True)  
 ax.set\_xlabel('Три відокремлений зразки')  
 ax.set\_ylabel('Спостережувані значення')



або тривимірну графіку

from mpl\_toolkits.mplot3d import Axes3D  
from matplotlib import cm  
from matplotlib.ticker import LinearLocator, FormatStrFormatter  
import numpy as np  
  
  
fig = plt.figure(figsize=(10, 10)) # ініціалізуємо об'єкт рисунок  
ax = fig.add\_subplot(projection='3d') # додаємо до об'єкту тривимірне представлення  
  
# ініціалізуємо дані  
X = np.arange(-5, 5, 0.25)  
Y = np.arange(-5, 5, 0.25)  
X, Y = np.meshgrid(X, Y) # заповнюємо поверхню значеннями по двом осям  
R = np.sqrt(X\*\*2 + Y\*\*2)  
Z = np.sin(R) # ініціалізуємо значення по вісі Oz  
  
# будуємо поверхню  
surf = ax.plot\_surface(X, Y, Z, cmap=cm.coolwarm) # будуємо тривимірну поверхню по заданим значенням  
  
  
ax.set\_zlim(-1.01, 1.01) # встановлюємо границі по вісі Oz  
ax.zaxis.set\_major\_locator(LinearLocator(10)) # встановлюємо 10 граничних ліній по вісі Oz  
ax.zaxis.set\_major\_formatter(FormatStrFormatter('%.02f')) # визначаємо формат виведення значень



## C.6 Інші можливості

Для Jupyter Notebook було створено велику кількість плагінів. Наприклад, можна вбудовувати відео з youtube:

from IPython.display import YouTubeVideo  
YouTubeVideo('kjBOesZCoqc')



Або інтерактивні карти (дана комірка відобразиться тільки якщо у вас встановлений [folium](https://github.com/python-visualization/folium). Якщо у вас нічого не відображається, то можете пропустити даний приклад, він далі не знадобиться)

Встановити необхідну бібліотеку можна через команду pip install назва бібліотеки, яку варто прописати в консолі, як представлено в прикладі нижче:



Або, як варіант, можна прописати команду прямо в комірці середовища Jupyter Notebook, як представлено в прикладі нижче:

!pip install folium

Requirement already satisfied: folium in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (0.14.0)  
Requirement already satisfied: requests in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (from folium) (2.27.1)  
Requirement already satisfied: numpy in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (from folium) (1.21.2)  
Requirement already satisfied: branca>=0.6.0 in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (from folium) (0.6.0)  
Requirement already satisfied: jinja2>=2.9 in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (from folium) (2.11.3)  
Requirement already satisfied: MarkupSafe>=0.23 in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (from jinja2>=2.9->folium) (2.0.1)  
Requirement already satisfied: charset-normalizer~=2.0.0 in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (from requests->folium) (2.0.4)  
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (from requests->folium) (1.26.9)  
Requirement already satisfied: idna<4,>=2.5 in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (from requests->folium) (3.3)  
Requirement already satisfied: certifi>=2017.4.17 in /Users/admin/opt/anaconda3/lib/python3.9/site-packages (from requests->folium) (2022.12.7)

import folium  
m = folium.Map(zoom\_start=12, location=[47.89829743895897, 33.36626740165739])  
m

<folium.folium.Map at 0x7fa025b867c0>

Або вбудувати будь-який інший шматок HTML за допомогою magic команди %%html. Нижче наведено приклад вбудовування посту з Твіттеру

%%html  
<blockquote class="twitter-tweet" data-lang="en"><p lang="en" dir="ltr">Replace &quot;AI&quot; with &quot;matrix multiplication &amp; gradient descent&quot; in the calls for &quot;government regulation of AI&quot; to see just how absurd they are</p>&mdash; Ben Hamner (@benhamner) <a href="https://twitter.com/benhamner/status/892136662171504640?ref\_src=twsrc%5Etfw">July 31, 2017</a></blockquote>  
<script async src="https://platform.twitter.com/widgets.js" charset="utf-8"></script>

<IPython.core.display.HTML object>

## C.7 Гарячі клавіші

Багато дій можна виконати за допомогою гарячих клавіш. Список гарячих клавіш можна знайти в меню *Help - Keyboard shortcuts*. Нижче наведено список найбільш корисних поєднань:

| Ключ | Описання |
| --- | --- |
| Esc | вийти з режиму редагування та виділити поточну комірку |
| Enter | перейти в режим редагування комірки поточної комірки |
| Ctrl+S, S | зберегти файл |
| Ctrl+Enter | виконати код і залишитися в поточній комірці |
| Shift + Enter | виконати код і перейти в наступну клітинку |
| Shift + Tab | виводить спливаюче вікно з документацією |
| a | додати комірку згори (above) |
| b | додати комірку знизу (below) |
| c | скопіювати комірку |
| v | вставити скопійовану клітинку |
| dd | видалити комірку |
| z | скасування останньої дії |