Міністерство освіти і науки України

Криворізький державний педагогічний університет

Південноукраїнський національний педагогічний університет імені К.Д. Ушинського

Київський національний економічний університет імені Вадима Гетьмана

Державний університет економіки і технологій

В. М. СОЛОВЙОВ, А. О. БЄЛІНСЬКИЙ

МОДЕЛЮВАННЯ СКЛАДНИХ СИСТЕМ У PYTHON

Навчально-методичний посібник
для самостійного вивчення дисципліни

Черкаси – 2024

УДК 330.4(075.8)
ББК 65в631я-1
С 60

Рекомендовано до друку рішенням Вченої ради
Криворізького державного педагогічного університету
(протокол № 5 від 09 листопада 2023 р.)

Рецензенти:

А.Ю.Ків, д.ф.-м.н., проф. (Південноукраїнський національний педагогічний університет імені К.Д.Ушинського)

А.В.Матвійчук, д.е.н., проф. (Київський національний економічний університет імені Вадима Гетьмана)

Ю.В.Триус, д.пед.н., проф. (Черкаський державний технологічний університет)

С 60 **Соловйов В.М., Бєлінський А.О.**
Моделювання складних систем у Python: Навчально-методичний посібник для самостійного вивчення дисципліни. – Черкаси: Видавець Третяков О.М, 2024. – 620 с.
ISBN 978-617-7827-70-1

У навчальному посібнику висвітлюються основні теоретичні та інструментальні аспекти моделювання складних систем різної природи з використанням інтерактивного веб-додатку Jupyter Notebook. Для дискретних даних у вигляді часових рядів розроблено і адаптовано сучасні міждисциплінарні методи кількісної оцінки складності: (мульти-)фрактальний, рекурентний, інформаційний, ентропійний та ін. види аналізу. Запропоновано відповідні міри складності. Показано, що більшість із них можна використовувати у якості індикаторів чи передвісників критичних або кризових явищ у складних системах.

Посібником зможуть скористатися як здобувачі другого (магістр) рівня вищої освіти спеціальності **014 Середня освіта (Фізика та астрономія; Інформатика; Математика; Природничі науки),** так і інших спеціальностей закладів вищої освіти, наукові працівники й особи, які цікавляться методами моделювання складних систем.

УДК 330.4(075.8)
ББК 65в631я-1

ISBN 978-617-7827-70-1

Зміст

Передмова

Не дивлячись на очікувані прогнози стати 21-му століттю століттям біології (за аналогією з 20-м століттям фізики), з легкої руки геніального британського астрофізика Стівена Гокінга (Stephen William Hawking, 1942-2018) його безумовно можна вважати століттям складності. Дійсно, виявилось, що складні системи різної природи проявляють схожі паттерни складності і можуть бути схарактеризовані універсальними міждисциплінарними кількісними методами і алгоритмами. Свідченням тому є Нобелівська премія з фізики 2021 р. "за новаторський внесок у наше розуміння складних фізичних систем" була присуджена італійському фізику Джорджіо Паризі "за відкриття взаємозв'язку безладу та флуктуацій у фізичних системах від атомного до планетарного масштабів" і японсько-американському кліматологу Сюкуро Манабе та німецькому вченому Клаусу Гассельманну "за фізичне моделювання клімату Землі, кількісну оцінку мінливості та надійне прогнозування глобального потепління".

Незважаючи на те, що визначення складних систем історично широко обговорювалося, навряд чи зараз існує широка згода щодо єдиного конкретного визначення: складна система утворена багатьма взаємодіючими елементами, які породжують явища, що виникають [1]. Однак при цьому залишаються невизначеними деякі терміни. Наприклад: скільки елементів достатньо для відображення складних емерджентних явищ? Цікаво, що ця кількість може бути великою, як кількість нейронів і синапсів у мозку людини, але також може бути відносно невеликою: мінімальна клітина також утворена лише кількома сотнями генів і вже жива. З цих прикладів стає зрозуміло, що справжня складність і нові явища вимагають ряду елементів, які можуть бути дуже далекими від великих чисел, які зазвичай розглядаються у фізиці (кількість молекул в молі газу визначається числом Авогадро, тобто приблизно 6×10^{23}). Здається дуже загальноприйнятою ідея, що складність виникає внаслідок конкуренції між випадковістю та порядком і що топологія складної системи за своєю суттю пов'язана з певною часткою випадковості в мережі взаємодій між її елементами (так званий, мережний підхід) або за знаком їх взаємодії (підхід спінового скла).

Точкою відліку, яку часто цитують у контексті колективних ефектів складних систем, є стаття "More is different", написана Філіппом Андерсоном (1923-2020) [2], лауреатом Нобелівської премії з фізики 1977 р. Наука про складні системи розглядає способи, якими складові частини породжують

колективну поведінку цілої системи. Однак виявилось, що така інтерпретація має обмежену корисність, оскільки охоплює занадто широкий набір обставин.

Історично деякий час тому у фізиці з'явився інший більш корисний крок у визначенні складних систем: система є складною, якщо її поведінка суттєво залежить від її деталей [3]. У цьому контексті маються на увазі такі явища, як детермінований хаос, квантова заплутаність, згортання білків, спінові стекла тощо. Колективна складна поведінка може виникати під впливом фрустрації та структурного безладу. Як наслідок, важко досягти стану рівноваги, реакції на зовнішні збурення повільні і дуже часто випадкові. Такі різні явища вивчаються в різних областях фізики: динамічні системи, квантова механіка та статистична фізика. Їх спільною рисою є те, що нескінченно малі зміни початкових умов (хоча й дуже різної природи) призводять до кардинально різних сценаріїв часової еволюції цих систем.

Є другий компонент, важливий для визначення складних систем. З одного боку, взаємодії між складовими частинами призводять до колективної поведінки та визначають макростан, але з іншого боку, взаємодії змінюються в ході еволюції системи та піддаються впливу макростану. Іншими словами, макростан і мікростани динамічно оновлюють один одного. Аналіз таких ефектів призвів до створення методів і розвитку концепцій, які були успішно застосовані для опису формально подібних явищ, що відбуваються в хімічних, біологічних, соціальних та інших системах, утворених агентами нефізичного походження.

Незважаючи на вражаючий прогрес науки про складність, якого було досягнуто за останні 50 років, ми все ще далекі від повного розуміння складності, оскільки ще не визначили необхідні умови для того, щоб система відображала складні емерджентні явища. Наприклад, ми досить далекі від повного розуміння роботи мозку. У результаті цього галузь може виглядати фрагментованою в порівнянні з іншими більш традиційними науковими галузями, наприклад, фізикою. Для вирішення проблеми складності нам дійсно потрібно досліджувати різні аспекти складних систем, і ми повинні прийняти відкриту точку зору, яка здатна описувати та прогнозувати дані складних систем, уникаючи попередньо визначених догм зверху вниз.

Наука про складність також є ключовою для розуміння та прогнозування еволюції великих пандемій та для інформування політиків та широкої громадськості про ризики поширення епідемії. Дійсно, мережна наукова спільнота вже задовго до COVID усвідомлювала небезпеку глобальних пандемій, які використовують переваги безмасштабних глобальних транспортних систем. На жаль, пандемія застала більшість країн зненацька,

оскільки плани на випадок надзвичайних ситуацій насправді не були готові до епідемії такого масштабу, як COVID-19. Для моніторингу розвитку цієї пандемії та будь-якої майбутньої пандемії вчені, ймовірно, поєднають велику кількість даних про соціальну мобільність із моделями прогнозування, які є ключовими для моніторингу пандемії та інформування політиків, незважаючи на багато невизначеностей щодо біологічної еволюції вірусів.

У майбутньому прогрес на стику між складністю та біологією стане ключовим для досягнення настільки необхідних успіхів у прецизійній медицині. Ця велика складна проблема вимагатиме справді міждисциплінарного підходу, який поєднує мережну науку, машинне навчання та штучний інтелект з молекулярною біологією та неврологією. Дійсно, в той час як біологія в останні десятиліття широко застосовувала одномолекулярний підхід або значною мірою покладалася на центральну догму молекулярної біології, тепер добре визнано, що більшість захворювань є складними, і для розуміння цих захворювань важливо прийняти складність і неоднорідність взаємодіючих мереж клітини. Нарешті, найближчим часом складність стане ключовою для створення фундаменту для квантового Інтернету, який вимагатиме поєднання прогресу квантової інформації з нашим розумінням класичних складних комунікаційних систем, таких як поточний Інтернет.

Іншим важливим викликом щодо надійності мережі є дослідження мозку, оскільки мозок, безсумнівно, є надійною складною системою, однак дуже важливо розуміти, як на його функцію впливають захворювання. Щоб відповісти на це питання, я вважаю, що нам потрібно прийняти стохастичну природу мозкової активності та отримати подальше розуміння взаємодії між функціями мозку та топологією мозкової мережі. Завдяки фундаментальним досягненням науки про мережі ми вже знаємо, що стійкість мереж до випадкових пошкоджень сильно залежить від статистичних властивостей мережі. Дійсно, безмасштабний розподіл ступенів мереж різко змінює фазову діаграму перколяції, демонструючи критичну поведінку, яка різко відрізняється від перколяції на регулярних решітках або на випадкових графах. Ці результати були ключовими для розуміння взаємодії між основною мережною структурою складних систем та їх динамікою.

Прогнозування складних систем є складним завданням і, звичайно, обмежене повсюдною нелінійністю їх динаміки. Проте за останні двадцять років було досягнуто важливого прогресу в прогнозуванні складних систем (наприклад, безпрецедентний прогрес у прогнозуванні поширення епідемії). Покращення потужності прогнозування складних систем здебільшого пов’язано

з великою кількістю даних, доступних для розробників моделей, і важливим прогресом, якого може досягти наука про складність у поєднанні з наукою про дані та штучним інтелектом (ШІ). Підтвердженням цьому є революційні зрушення у матеріалознавстві [4] чи фінансах [5].

Підвищення наших можливостей прогнозування складних систем, які врешті-решт поєднають мережну науку, науку про дані та алгоритми штучного інтелекту, є справді ключовим для різноманітних застосувань, зокрема для забезпечення можливих сценаріїв зміни клімату. Однак потужність простих моделей для розуміння складних систем має надзвичайно важливе значення для інтерпретації результатів: прості моделі можуть не охопити всі деталі складних систем, але дозволяють зрозуміти та приборкати складність, що може мати вирішальне значення при розробці кращих алгоритмів ШІ. Теорія спінового скла вчить нас, що модель, яка насправді є досить простою (лише додавання випадкової суміші позитивних і негативних взаємодій до моделі Ізінга в повністю зв’язній мережі), може бути вже дуже складною.

Методам моделювання складних систем присвячені наші попередні монографії та навчально-методичні посібники, орієнтовані на систему комп’ютерної математики Матлаб [6–9]. З урахуванням домінування Python у прикладних дослідженнях складних систем поява даного посібника на думку авторів є виправданою.

Дана онлайн-книга буде часто оновлюватись і редагуватись. Її вміст є вільним для використання. Метою авторів даної книги було забезпечення:

- доступності вивчення моделювання складних систем різної природи та складності;
- можливість постійого оновлення даної книги, що дозволяє динамічний характер покращення мови програмування Python і web-інтерактивного середовища Jupyter Notebook.

Читачам рекомендується посилатися як на цю книгу, так і на дотичні до неї джерела, коли вони використовують викладені тут методології та ідеї. При цитуванні слід дотримуватися точності деталей, особливо URL-адреси, щоб спрямувати читачів до правильної Інтернет сторінки.

Нижче наведено один із варіантів цитування:

> Соловйов В. М., Бєлінський А. О. Моделювання складних систем у Python: Навчально-методичний посібник для самостійного вивчення дисципліни. GitHub. Черкаси: Видавець О.М. Третяков. URL: https://butman2099.github.io/Complex-systems-book/ (дата звернення: 15.05.2024).

Посилання

1. Bianconi G. et al Complex systems in the spotlight: next steps after the 2021 Nobel Prize in Physics (2023) J. Phys. Complex. 4 010201 https://iopscience.iop.org/article/10.1088/2632-072X/ac7f75
2. Anderson P.W. More is different. New Series, Vol. 177, No. 4047. (Aug. 4, 1972), pp. 393-396. https://cse-robotics.engr.tamu.edu/dshell/cs689/ papers/ anderson72more_is_different.pdf
3. Holovatch Y., Kenna R., Thurner S. Complex systems: physics beyond physics (2017) Eur. J. Phys. 38 023002. https://doi.org/10.1088/1361-6404/aa5a87
4. Merchant A. et al. Scaling deep learning for materials discovery (2023) Nature. Vol 624 (7 December 2023), pp.80-90. https://doi.org/10.1038/s41586-023-06735-9
5. Chu, Z., Guo, H., Zhou, X., Wang, Y., Yu, F., Chen, H., Xu, W., Lu, X., Cui, Q., Li, L. and Zhou, J., 2023. Data-centric financial large language models. arXiv preprint arXiv:2310.17784. https://doi.org/10.48550/arXiv.2310.17784
6. Соловйов В. М. Математична економіка: навч.-метод. посіб. для самостійного вивч. / В. М. Соловйов. – Черкаси: Видавництво ЧНУ ім. Б. Хмельницького, 2008. – 136 с. http://elibrary.kdpu.edu.ua/handle/0564/1049https://doi.org/10.31812/0564/1049
7. Дербенцев В. Д. Синергетичні та економфізичні методи дослідження динамічних та структурних характеристик економічних систем: монографія / В. Д. Дербенцев, О. А. Сердюк, В. М. Соловйов, О. Д. Шарапов. – Черкаси: Брама-Україна, 2010. – 300 с. http://elibrary.kdpu.edu.ua/handle/0564/1045https://doi.org/10.31812/0564/1045
8. Ганчук А. А. Методи прогнозування: навчальний посібник / А. А. Ганчук, В. М. Соловйов, Д. М. Чабаненко. – Черкаси: Брама-Україна, 2012. – 140 с. http://elibrary.kdpu.edu.ua/handle/0564/1186https://doi.org/10.31812/0564/1186
9. Соловйов В. М. Моделювання складних систем : навчально-методичний посібник для самостійного вивчення дисципліни / В. М. Соловйов, О. А. Сердюк, Г. Б. Данильчук. – Черкаси: Видавець О. Ю. Вовчок, 2016. – 204 с. http://elibrary.kdpu.edu.ua/handle/0564/1065https://doi.org/10.31812/0564/1065

1. Лабораторна робота № 1

Тема. Аналіз флуктуацій часового ряду

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

1.1 Теоретичні відомості

1.1.1 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.

1.1.2 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 T, що є рухомим параметром нашої оцінки*. Для індексу $X(t)$ визначимо зміну ціни $G(t)$ як зміну логарифмів індексів:

$$G(t) = \ln X(t + \Delta t) - \ln X(t) \cong [X(t + \Delta t) - X(t)]/X(t), \qquad (1.1)$$

де Δt є часовим інтервалом затримки. Величину (1.1) називають прибутковістю (return). Якщо використовувати границі, то малі зміни $X(t)$ приблизно відповідають змінам, визначеним другою рівністю. Ми лише підраховуємо час роботи ринку, викидаємо ночі, вихідні та свята із набору даних, тобто, вважаємо, що ринок працює без перерв.

Модуль $G(t)$ описує амплітуду флуктуацій. У порівнянні зі значеннями $G(t)$ їх модуль не показує глобальних трендів, але великі значення $G(t)$ відповідають крахам та великим миттєвим змінам на ринках.

Визначимо волатильність як середнє від $G(t)$ для часових вікон $T = n \cdot \Delta t$, тобто

$$V_T = \frac{1}{n} \sum_{t'=t}^{t+n-1} |G(t')|, \qquad (1.2)$$

де n є цілим числом. Таке визначення може бути ще узагальнене заміною $G(t)$ на $|G(t)|^\gamma$, де $\gamma > 1$ дає більш виражені великі значення $G(t)$, в той час як $0 < \gamma < 1$ виділяє малі значення $G(t)$.

У цьому визначенні волатильності використовується два параметри: Δt та n. Параметр n є шаблонним (чи модельним) часовим інтервалом для даних, а параметр Δt є кроком переміщення часового вікна. Зауважимо, що вказане визначення волатильності має внутрішню помилку, а саме: вибір більшого часового інтервалу T веде до збільшення точності визначення волатильності.

Однак, велике значення T також включає погане розбиття часу на інтервали, що веде, у свою чергу, до врахування не всієї прихованої у ряді інформації.

1.1.3 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 N пар спостережень двох змінних x та y. Коефіцієнт кореляції між парами x та y визначається як

$$r = [\sum(x_i - \bar{x})(y_i - \bar{y})] \Big/ \left[\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}\right] \tag{1.3}$$

де сума знаходиться за всіма N 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. Отже, автокореляція першого порядку використовує перші $N-1$ спостережень $x_t, t = 1, \ldots, N-1$ та наступні $N-1$ спостережень $x_t, t = 2, \ldots, N$.

Кореляція між x_t та $x_t + 1$ визначається як

$$r_1 = \left[\sum_{t=1}^{N-1}(x_t - \bar{x})\,(x_{t+1} - \bar{x})\right] \Big/ \left[\sum_{t=1}^{N}(x_t - \bar{x})^2\right], \tag{1.4}$$

де x — це середнє для досліджуваного періоду.

Рівняння (1.4) може бути узагальнене для отримання кореляції між спостереженнями, розділеними k часовими інтервалами:

$$r_k = \left[\sum_{t=1}^{N-k} (x_t - \bar{x})\,(x_{t+k} - \bar{x})\right] \Big/ \left[\sum_{t=1}^{N} (x_t - \bar{x})^2\right]. \qquad (1.5)$$

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

1.2 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! 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.

1.2.1 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
msft.info
```

Можна вилучити ринкові значення за максимальний період часу:

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

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

```
# показувати дії (дивіденди, спліти)
msft.actions

# продемонструвати дивіденти
msft.dividends

# продемонструвати спліти
msft.splits
```

Для методу `history()` доступні 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).

1.2.2 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
```

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

```
data = yf.download("SPY AAPL",
                   start="2017-01-01",
                   end="2017-04-30",
                   group_by="ticker")
```

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 нас ще будуть цікавати 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 з числовими таблицями та часовими рядами;
- **`numpy`**: бібліотека, що додає підтримку великих багатовимірних масивів і матриць, а також 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. Встановимо 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

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                   # розмір фонтів рисунку
'lines.linewidth': 2,                # товщина ліній
'axes.titlesize': 'small',           # розмір титулки над рисунком
'axes.labelsize': size,              # розмір підписів по осям
'legend.fontsize': size,             # розмір легенди
'xtick.labelsize': size,             # розмір розмітки по осі Ox
'ytick.labelsize': size,             # розмір розмітки по осі Ox
"font.family": "Serif",              # сімейство стилів підписів
"font.serif": ["Times New Roman"],   # стиль підпису
'savefig.dpi': 300,                  # якість збережених зображень
'axes.grid': False                   # побудова сітки на самому рисунку
}

plt.rcParams.update(params)          # оновлення стилю згідно налаштувань
```

Представлені налаштування є орієнтовними і можуть змінюватись у ході наступних лабораторних роботах. 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 Ox
ylabel = symbol                             # підпис по вісі Oy
```

> **Важливо**
>
> Представлені в подальшому методи є універсальними. Іншими словами, ви можете їх використовувати не лише для фінансових часових рядів, але й для біологічних, фізичних та 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 твердого тіла. Представлену далі залежність було отримано в результаті механічних випробувань певного металу. Зрозуміло, що аналіз результатів і висновки у цьому випадку залежать від того, з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'    # підпис по вісі Ox
ylabel = symbol              # підпис по вісі Oy
```

Тепер виведемо значення, що були зчитані з текстового документа:

```
fig, ax = plt.subplots(1, 1)              # 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 Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')              # Зберігаємо графік
plt.show();# Виводимо графік
```

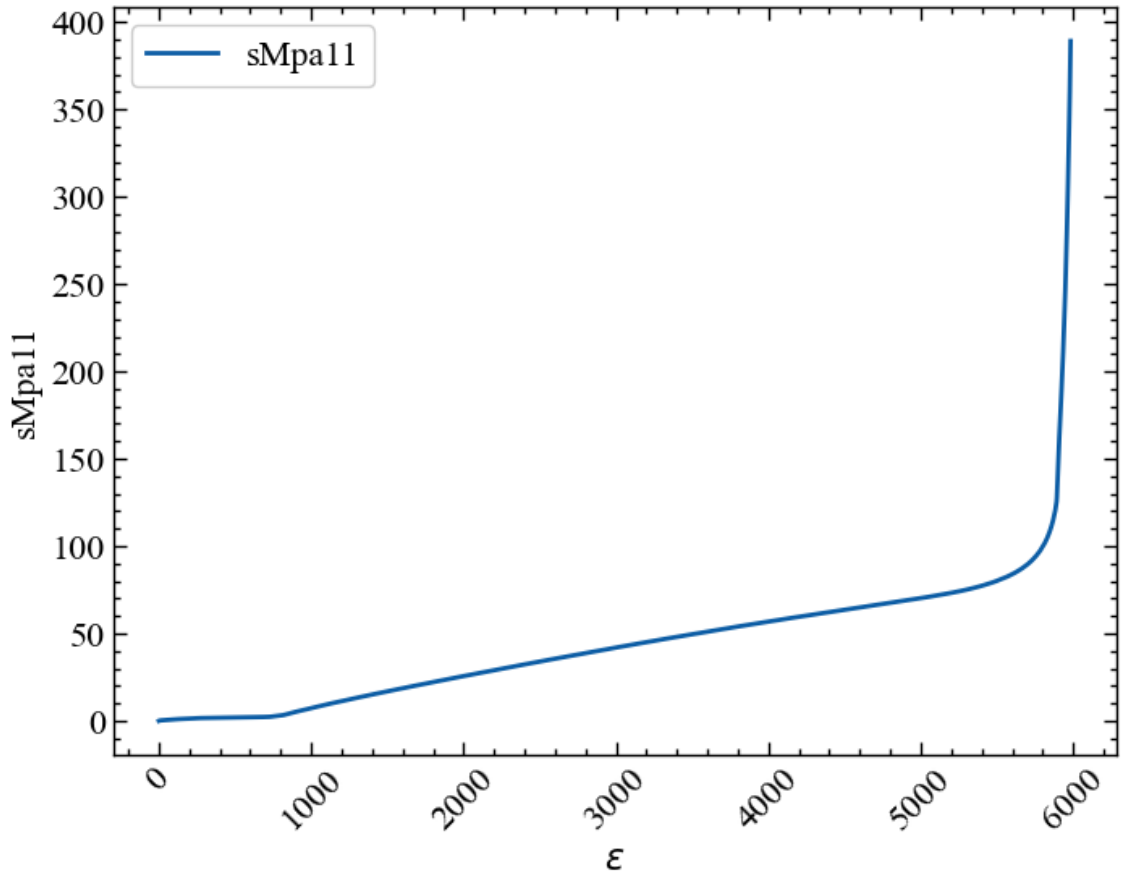


Рис. 1.1: Діаграма деформації

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(1, 1)                # 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 Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();# Виводимо графік
```

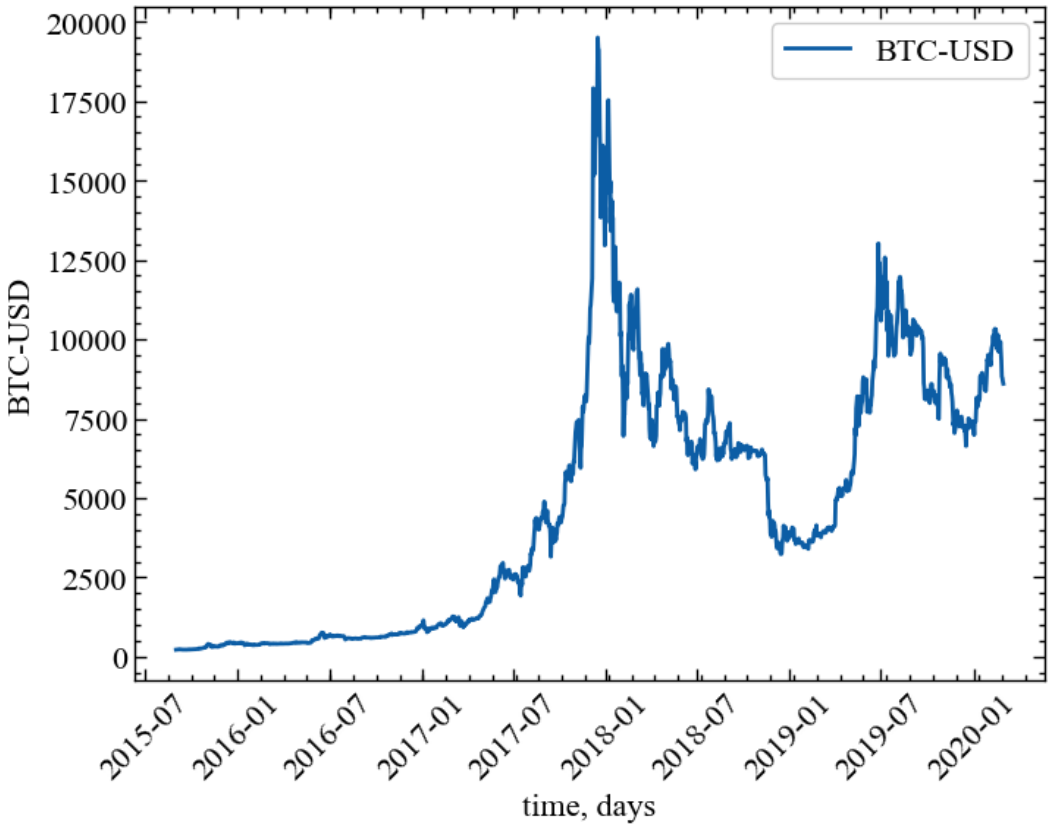


Рис. 1.2: Динаміка щоденних змін індексу Біткоїна

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

Прибутковості розраховуватимуться згідно рівняння (1.1). У Python для цього ми використовуватимемо метод `pct_change()`, що доступний нам завдяки бібліотеці `pandas`.

Стандартизовані прибутковості можна визначити як $g(t) = [G(t) - \mu]/\sigma$, де μ відповідає середньому значенню прибутковостей за досліджуваний часовий інтервал, а σ 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 Ox
ax.set_ylabel(ylabel + ' прибутковості')     # Додаємо підпис для вісі Oy
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 розмежує -3 сигма події

plt.xticks(rotation=45)                      # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'Прибутковості{symbol}.jpg')  # Зберігаємо графік
plt.show();                                  # Виводимо графік
```

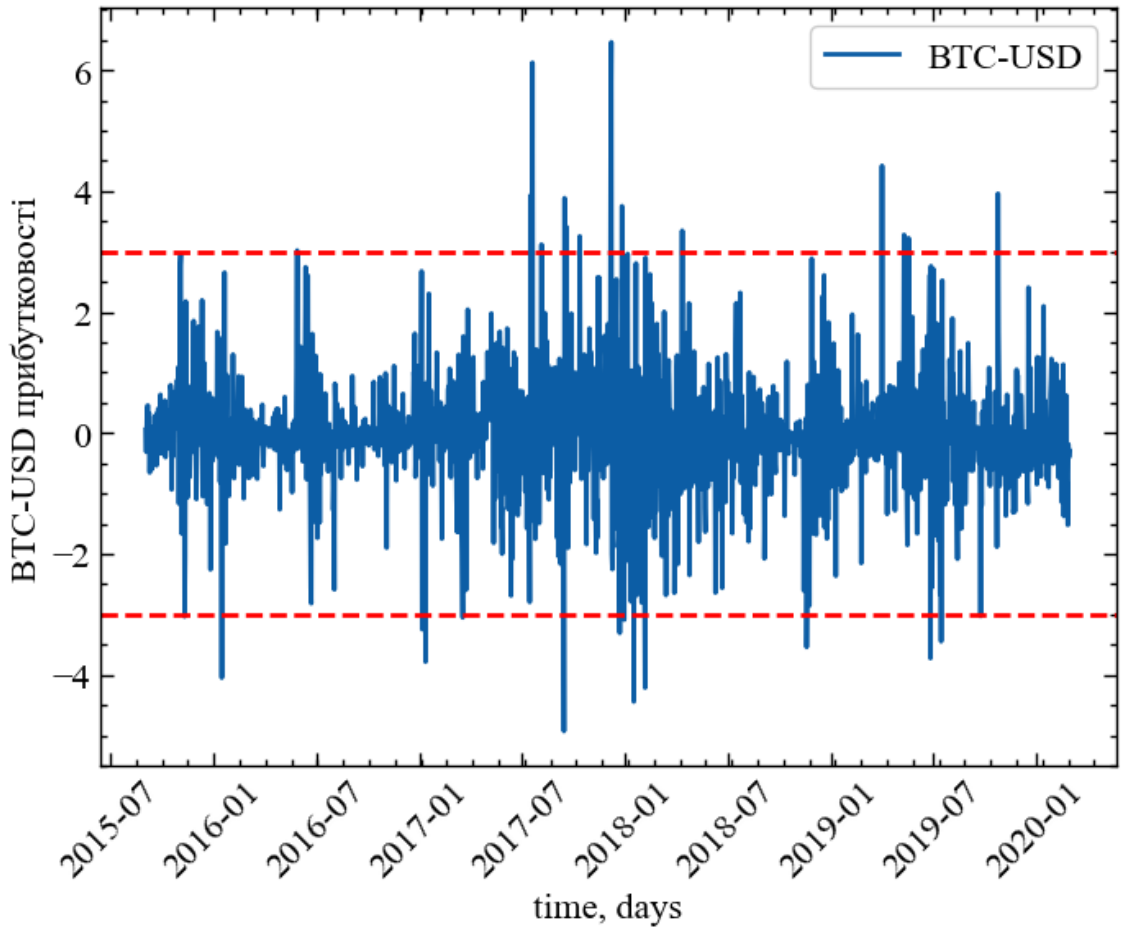


Рис. 1.3: Нормалізовані прибутковості досліджуваного часового ряду

Зверніть увагу, що флуктуації нормалізованих прибутковостей досить часто перевищують величину $\pm 3\sigma$, що, як відомо, надзвичайно рідко спостерігається для незалежних подій. Цей факт можна відобразити шляхом порівняння функції розподілу нормалізованих флуктуацій з розподілом Гауса (Рис. 1.4). 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 за аналогією 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 `norm` для дійсних значень x має наступний вигляд: $f(x) = \exp(-x^2/2)/\sqrt{2\pi}$.

```
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(1, 1)                 # Створюємо порожній графік
ax.plot(x, p, label='Гаусіан')               # 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 Ox
ax.set_ylabel(r"$f(g)$")                     # Додаємо підпис для вісі Oy
ax.set_yscale('log')

plt.savefig(f'Гаус + прибутковості {symbol}.jpg')   # Зберігаємо графік
plt.show();                                         # Виводимо графік
```

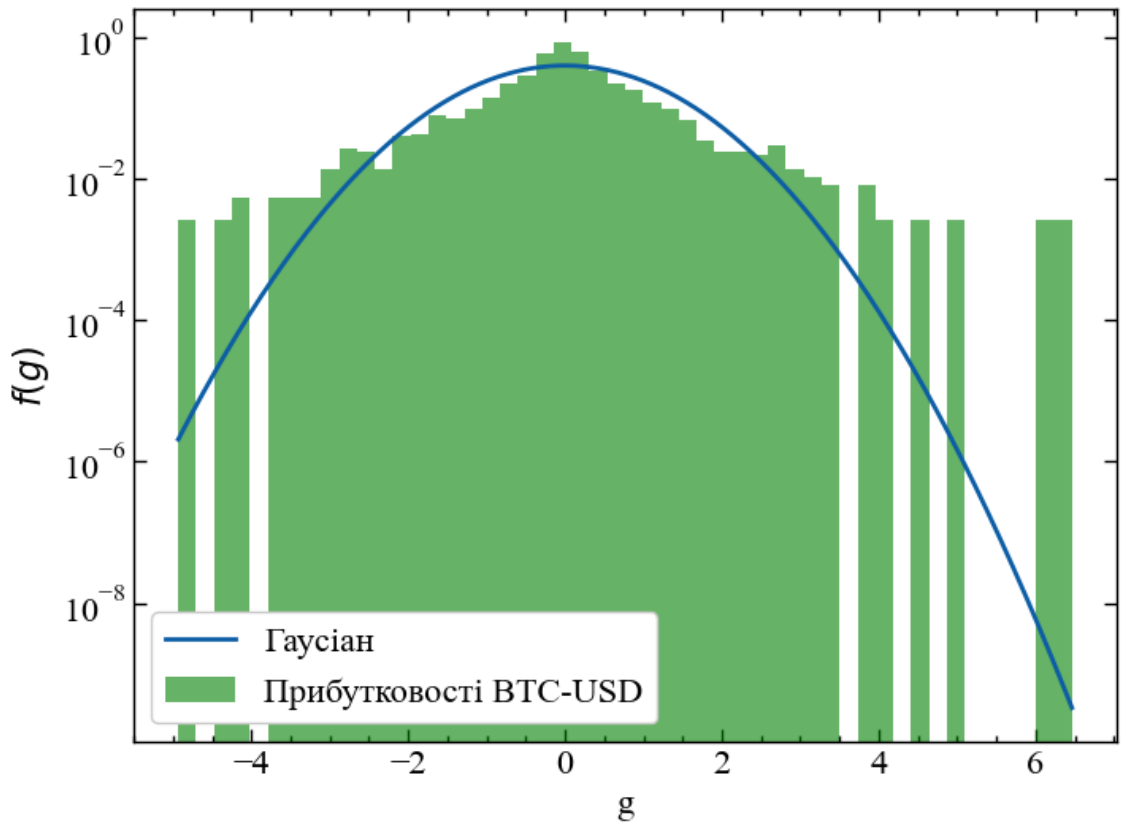


Рис. 1.4: Порівняння функцій розподілу нормалізованих прибутковостей з нормальним розподілом

Як ми можемо бачити, крива Гауса відхиляється від істинної частоти настання подій, що перевищують $\pm 3\sigma$, і ми можемо стверджувати, що 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` швидшого алгоритму. Інші методи зберігаються з причин застарілості, але не рекомендуються. Вони 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 із 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(1, 1)                # 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(r"Часовий лаг $k$")           # Додаємо підпис для вісі Ox
ax.set_ylabel(r"Автокореляція $r$")         # Додаємо підпис для вісі Oy
ax.set_ylim(-1.1, 1.1)                      # Встановлюємо обмеження по вісі Oy

plt.savefig(f'Автокореляції {symbol}.jpg')  # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

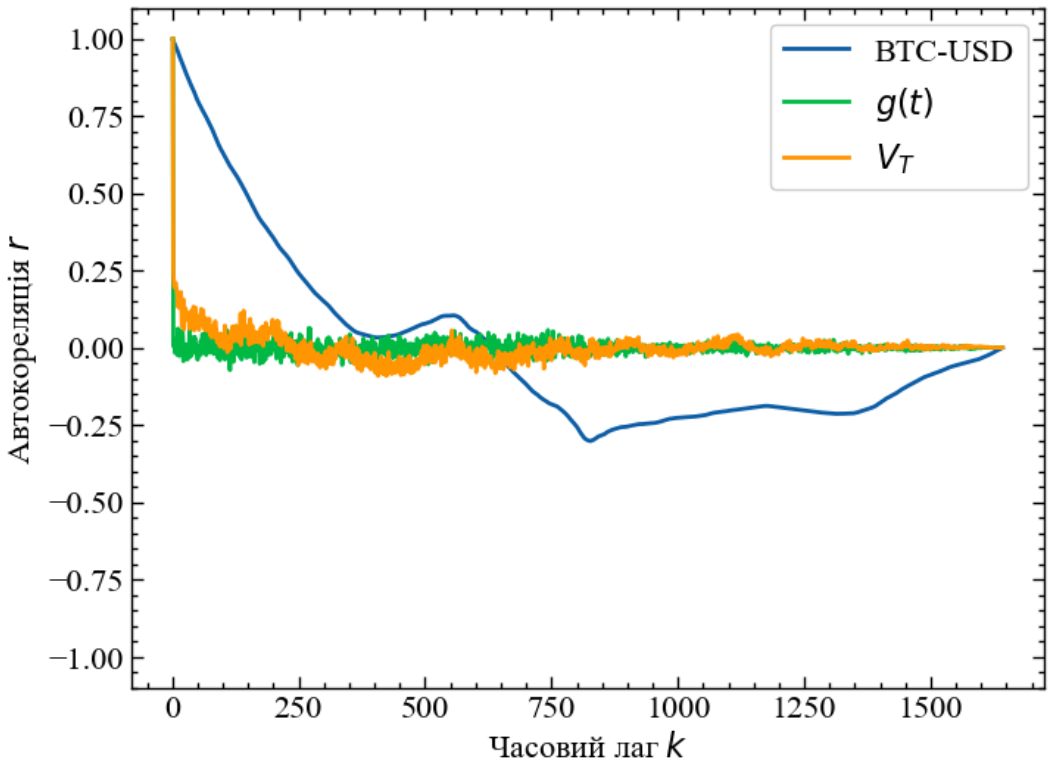


Рис. 1.5: Зміна з часом парних автокореляційних функцій для вихідного ряду x, нормалізованих прибутковостей g та їх модулів $|g|$

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 подальші розрахунки варто виконувати не для всього ряду, а для його фрагментів.

Сформулюємо алгоритм **ковзного (рухомого) вікна**. Виділимо фрагмент часового ряду (вікно), розрахуємо необхідні міри складності, змістимо вікно вздовж часового ряду на заздалегідь визначену величину (крок) і повторимо процедуру до вичерпання часового ряду. 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 характерним чином для всіх особливих періодів (наприклад, крахів), зменшується або збільшується у передкризовий період, то вона може служити їх індикатором або передвісником.

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])
```

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 знаходили для рухомого вікна. Результати представлено на Рис. 1.6.

```
fig, ax = plt.subplots(1, 1)

ax2 = ax.twinx()
ax3 = ax.twinx()

ax2.spines.right.set_position(("axes", 1.03))
ax3.spines.right.set_position(("axes", 1.10))

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],
               autocorr, "g-", label=r"$A$")

p3, = ax3.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())

tkw = dict(size=4, width=1.5)
ax.tick_params(axis='y', colors=p1.get_color(), **tkw)
ax2.tick_params(rotation=90, axis='y', colors=p2.get_color(), **tkw)
ax3.tick_params(rotation=30, axis='y', colors=p3.get_color(), **tkw)

ax.tick_params(rotation=45, axis='x', **tkw)

ax3.legend(handles=[p1, p2, p3], fontsize=18)
```

```
plt.savefig(f"all_name={symbol}_ret={ret_type}_\
            wind={window}_step={tstep}.jpg")

plt.show();
```

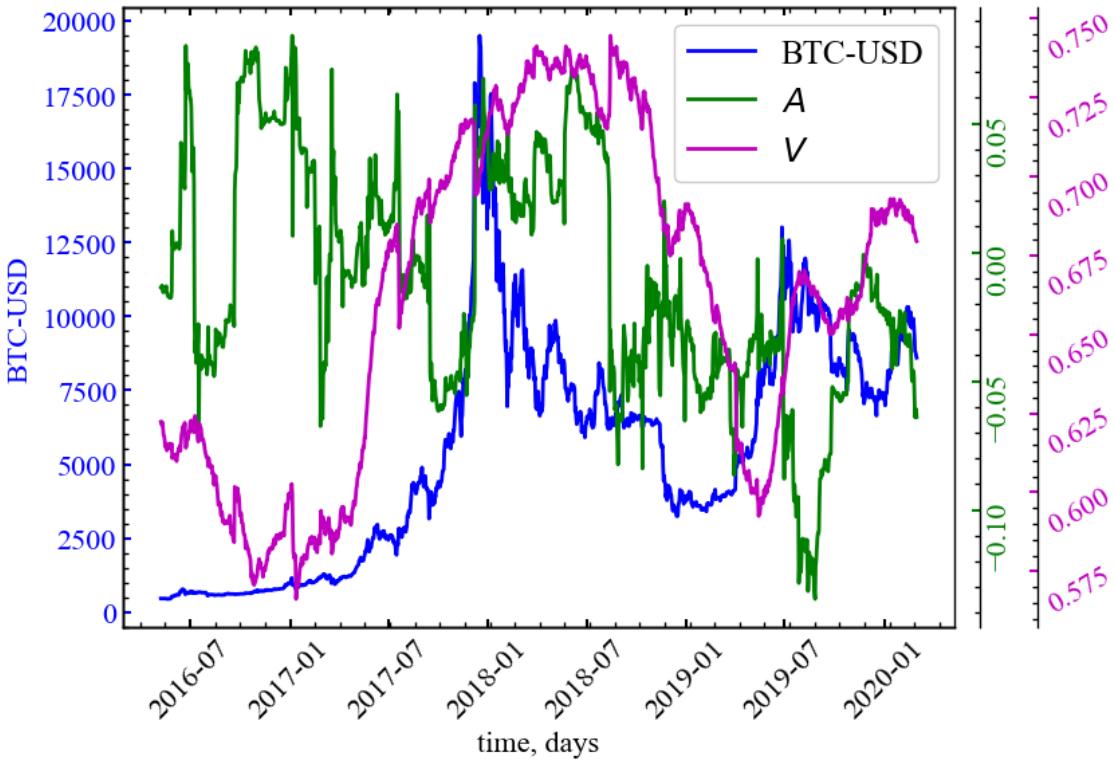


Рис. 1.6: Динаміка індексу Біткоїна, віконних автокореляції та волатильності

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 наслідком збурень у процесі роботи ринку. Аналіз таких збурень, їх частоти та сили, дозволяє виявляти приховані закономірності роботи ринку.

1.3 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 надавати корисні рекомендації фінансовим аналітикам.

1.4 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

1.5 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?

1.6 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. За вказаним посиланням та номером у списку групи оберіть компанію, що входить до індексу та проведіть відповідні розрахунки. Порівняйте отримані результати з такими ж для Біткоїна.

2. 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

2.1 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 в природний спосіб. 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 у 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 опишемо теорію рекурентного аналізу, наведемо деякі приклади, розглянемо його можливі області застосування при аналізі і прогнозування складних фінансово-економічних систем.

2.1.1 Фазовий простір та його реконструкція

Стан системи описується її змінними стану

$$x^1(t), x^2(t), \ldots, x^d(t),$$

де верхній індекс — номер змінної. Набір із d змінних стану в момент часу t складає вектор стану $\vec{x}(t)$ в d-вимірному фазовому просторі. Даний вектор переміщується в часі та в напрямі, що визначається його вектором швидкості:

$$\dot{\vec{x}}(t) = \partial_t \vec{x}(t) = \vec{F}(t).$$

Послідовність векторів $\vec{x}(t)$ утворює траєкторію у фазовому просторі, причому поле швидкості $\vec{F}$ дотичне до цієї траєкторії. Еволюція траєкторії описує динаміку системи і її атрактор. Знаючи $\vec{F}$, можна одержати інформацію про стан системи в момент t 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 відсутня інформація щодо всіх змінних стану, або не всі з них можливо виміряти. Як правило, маємо єдине спостереження, проведене через дискретний часовий інтервал Δt. Таким чином, вимірювання записуються у вигляді ряду $u_i(t)$ і, де $t = i \cdot \Delta t$. Інтервал Δt може бути постійним, проте це не завжди можливо і створює проблеми для застосування стандартних методів аналізу даних, що вимагають рівномірної шкали спостережень.

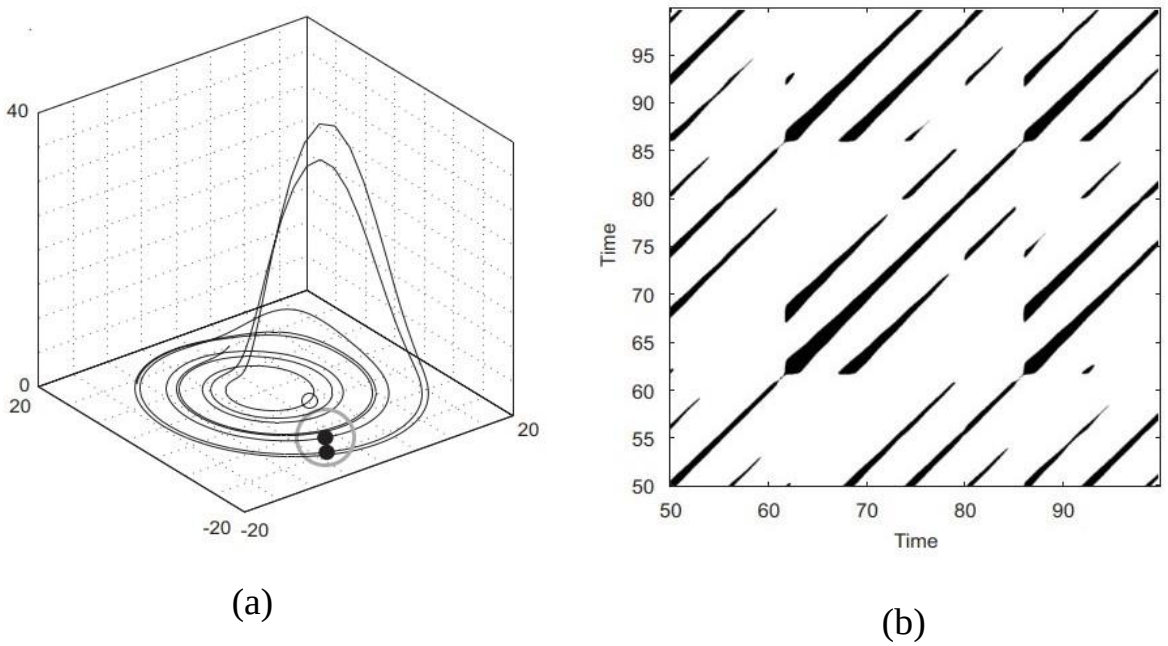
Взаємодії і їх кількість у складних системах такі, що навіть за однією змінною стану можна судити про динаміку всієї системи в цілому (даний факт був встановлений групою американських учених при вивченні турбулентності). Таким чином, еквівалентна фазова траєкторія, що зберігає структури оригінальної фазової траєкторії, може бути відновлена з одного спостереження або часового ряду [1] за теоремою Такенса (Takens) методом **часових затримок** [2]:

$$\hat{\vec{x}}(t) = \left(u_i, u_{i+\tau}, \ldots, u_{i+(d_E-1)\tau}\right).$$

Тут d_E — розмірність вкладення, τ — часова затримка (реальна часова затримка визначається як $\tau \cdot \Delta t$). Топологічні структури відновленої траєкторії зберігаються, якщо $d_E \geq 2 \cdot d + 1$, де d — розмірність атрактора [2]. На практиці у більшості випадків атрактор може бути відновлений і при $d_E \leq 2d$. Затримка, як правило, вибирається апріорно.

Існує кілька підходів до вибору мінімально достатньої розмірності m, 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 розмірності. Існують і інші методи, засновані на цій концепції, наприклад, визначення відносин відстаней між одними і тими ж сусідніми точками при різних d_E. Розмірність атрактора також може бути визначена за допомогою крос-кореляційних сум.



(a) (b)

Рис. 2.1: Відрізок траєкторії у фазовому просторі системи Рьослера i (a) та відповідний рекурентний графік (b). Вектор фазового простору в точці j, який потрапляє в околицю (сіре коло в (a)) заданого вектора фазового простору вектора в точці i вважається точкою рекурентності (чорна точка на траєкторії в (a)). Вона позначається чорною точкою на рекурентній діаграмі у позиції (i, j). Вектор фазового простору за межами околу (порожнє коло в (a)) позначається білою точкою рекурентній діаграмі

2.1.2 Рекурентний аналіз

Процесам у природі властива яскраво виражена рекурентна поведінка, така, як періодичність або іррегулярна циклічність. Більш того, рекурентність (повторюваність) станів у 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 вона 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, навіть мале збурення в складній динамічній системі може привести систему до експоненціального відхилення від її стану, через деякий час система прагне повернутися до стану 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 m-вимірної фазової траєкторії станів системи $\vec{x}(t)$ завдовжки N на двовимірну квадратну двійкову матрицю розміром $N \times N$ [3], в якій 1 (чорна точка) відповідає повторенню стану при деякому часі i в деякий інший час j, 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

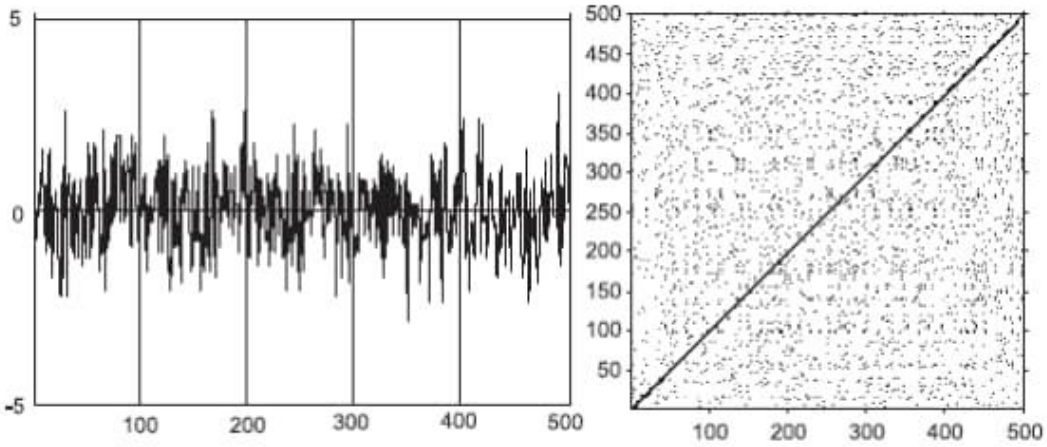
$$R_{i,j}^{d_E,\varepsilon_i} = \Theta\left(\varepsilon_i - \left\|\vec{x}_i - \vec{x}_j\right\|\right), \quad \vec{x} \in \Re^{d_E}, \quad i,j = 1,\dots,N,$$

де N — кількість даних станів, x_i, ε_i — розмір околиці точки $\vec{x}$ у момент i, $\|\cdot\|$ — норма і $\Theta(\cdot)$ — функція Хевісайда.

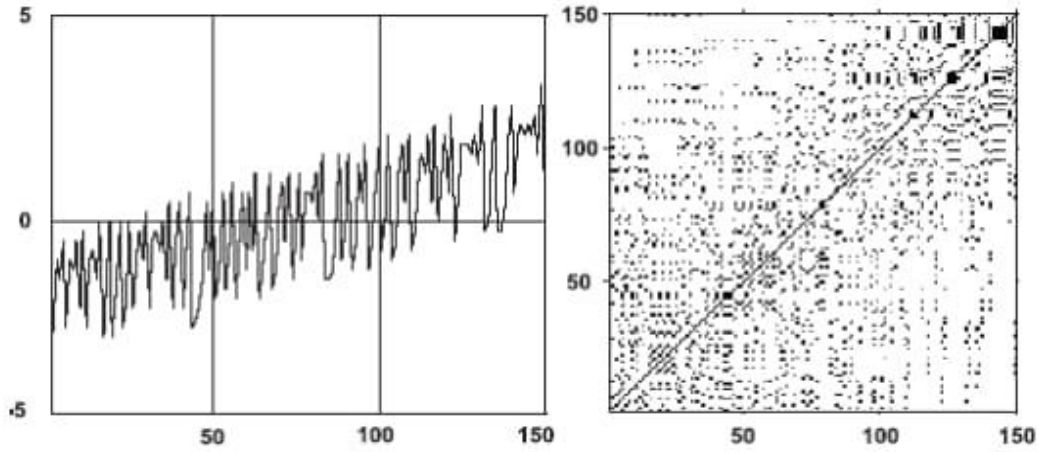
Непрактично і, як правило, неможливо знайти повну рекурентність у значенні $\vec{x}_i \equiv \vec{x}_j$ (стан динамічної, а особливо — хаотичної системи не повторюється повністю еквівалентно початковому стану, а підходить до нього скільки завгодно близько). Таким чином, рекурентність визначається як достатня близькість стану $\vec{x}_j$ до стану $\vec{x}_i$. Іншими словами, рекурентними є стани $\vec{x}_j$, які потрапляють в m-вимірну околицю з радіусом ε_i і центром в $\vec{x}_i$. Ці точки $\vec{x}_j$ називаються **рекурентними точками** (recurrence points) [4,5].

Оскільки $R_{i,i} = 1,\ i = 1,\ldots,N$ за визначенням, то рекурентна діаграма завжди містить чорну діагональну лінію — **лінію ідентичності** (line of identity, LOI) під кутом $\pi/4$ до осей координат. Довільно узята рекурентна точка не несе якої-небудь корисної інформації про стани в часи i і j. Тільки вся сукупність рекурентних точок дозволяє відновити властивості системи.

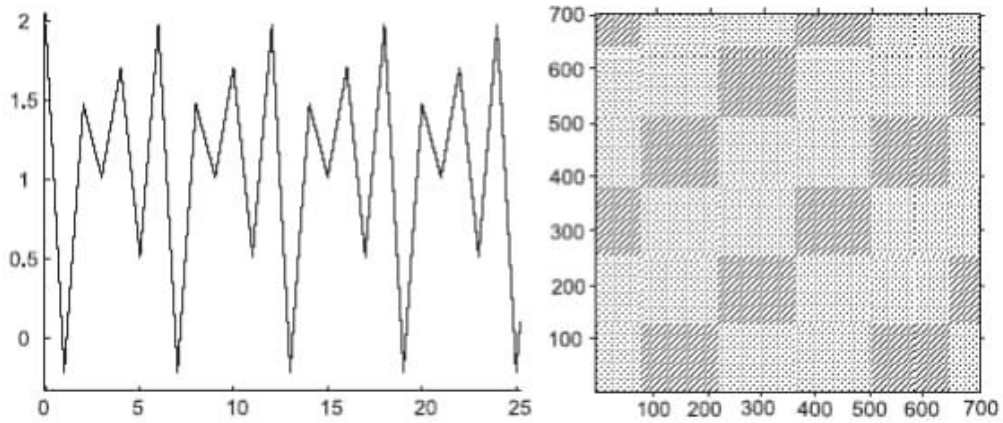
Зовнішній вигляд рекурентної діаграми дозволяє судити про характер процесів, які протікають в системі, наявності і впливі шуму, станів повторення і завмирання (ламінарності), здійсненні в ході еволюції системи різких змін стану (екстремальних подій).



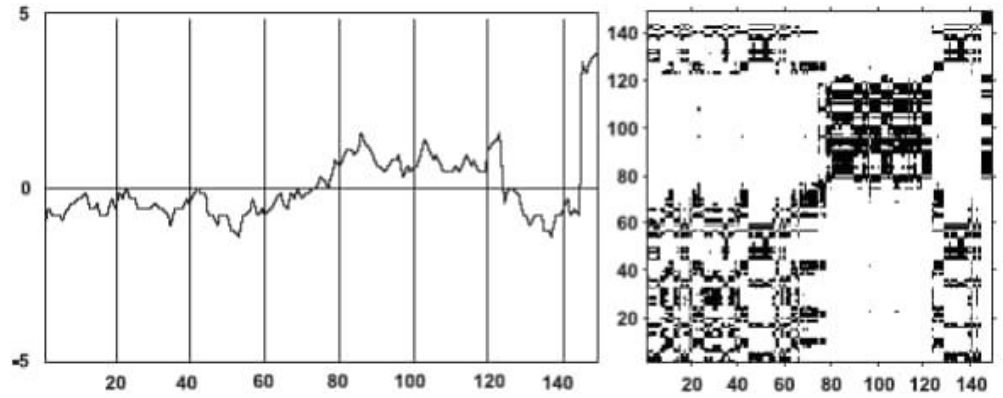
(a)



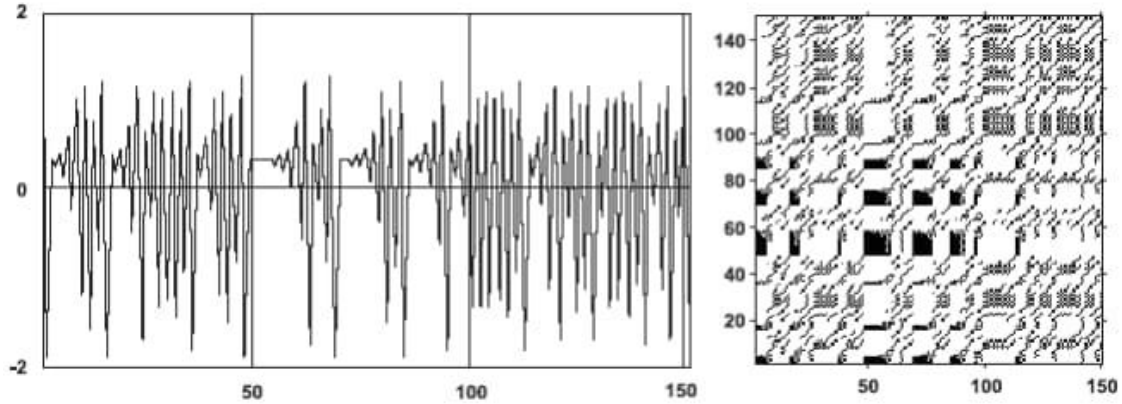
(b)



(c)



(d)



(e)

Рис. 2.2: Динамічні ряди, що характеризують однорідність (a), дрейф (b), осциляції (c), контрастну топологію (d), ламінарність (e) та побудовані для них рекурентні діаграми

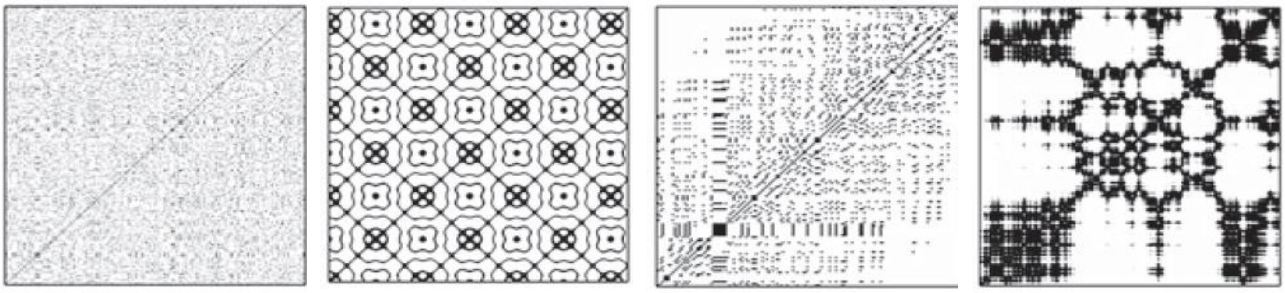
2.1.3 Аналіз діаграм

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, патерни у шаховому порядку) відповідають різним осцилюючим системам з періодичністю в динаміці;
- **дрейф** відповідає системам з параметрами, що поволі змінюються, і це 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.



(a) (b) (c) (d)

Рис. 2.3: Характерні топології рекурентних діаграм: (a) — однорідна (нормально розподілений шум); (b) — періодична (генератор Ван дер Поля); (c) — дрейф (відображення Ікеди з накладеною послідовністю, що лінійно росте); (d) — контрастні області або смуги (узагальнений броунівський рух) [6]

Детальний 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*** $R_{i+k,j+k} = 1$ (при $k = 1 \ldots l$ де l — 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 $\pi/4$ 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*** $R_{i,j+k} = 1$ (при $k = 1 \ldots v$, де v — довжина вертикальної або горизонтальної лінії) виділяють проміжки часу, в котрі стан системи не змінюється або змінюється не суттєво (система як би "заморожена" на цей час), що є ознакою "ламінарних" станів.

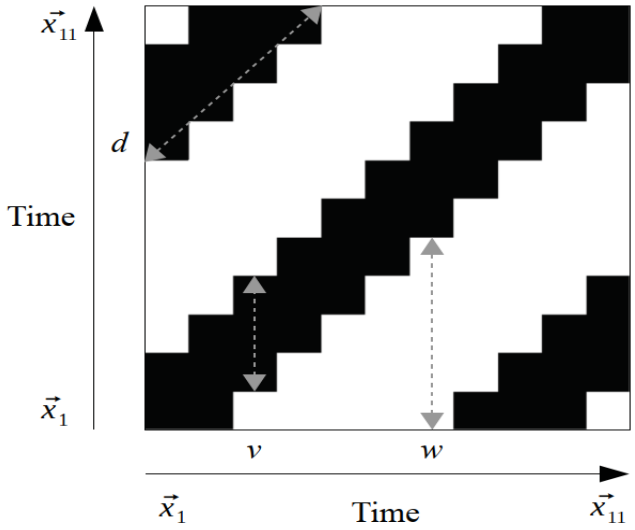


Рис. 2.4: Основні концепції рекурентного аналізу. Відображена діаграма рекурентності базується на часовому ряді, що було реконструйовано до 11 реконструйованих векторів, від $\vec{X}(0)$ до $\vec{X}(10)$. Виділено діагональну лінію довжиною $d = 3$, вертикальну лінію довжиною $v = 3$ і білу вертикальну лінію довжиною $w = 5$ [7].

2.2 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.

2.2.1 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;
- **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. Він базується на ідеї, що динамічна система може бути описана вектором чисел, який називається її “станом”, і має на меті забезпечити повний

опис системи в даний момент часу. Множина всіх можливих станів називається “простором станів”.

Теорема Такенса [2] припускає, що послідовність вимірювань динамічної системи містить у собі всю інформацію, необхідну для повної реконструкції простору станів. Метод часових затримок намагається визначити стан s системи в певний момент часу t, 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 d_E-розмірні реконструйовані часові вкладення

$$\vec{x}_i = \left(x_i, x_{i+\tau}, \ldots, x_{i+(d_E-1)\tau}\right), \tag{2.1}$$

а i проходить в діапазоні $1, \ldots, N-(d_E-1)\tau$; значення τ представляє часову затримку, а d_E — 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:

`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 τ іноді називають запізненням). Ще розглянемо метод `complexity_delay()` для оцінки оптимального значення цього параметра;
- **dimension** (*int*) — розмірність вкладень (d_E, іноді позначається як d або m). 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
import pandas as pd

%matplotlib inline
```

І виконаємо налаштування рисунків для виводу:

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                   # розмір фонтів рисунку
'lines.linewidth': 2,                # товщина ліній
'axes.titlesize': 'small',           # розмір титулки над рисунком
'axes.labelsize': size,              # розмір підписів по осям
'legend.fontsize': size,             # розмір легенди
'xtick.labelsize': size,             # розмір розмітки по осі Ox
'ytick.labelsize': size,             # розмір розмітки по осі Ox
"font.family": "Serif",              # сімейство стилів підписів
"font.serif": ["Times New Roman"],   # стиль підпису
'savefig.dpi': 300,                  # якість збережених зображень
'axes.grid': False                   # побудова сітки на самому рисунку
}
```

```
plt.rcParams.update(params)                # оновлення стилю згідно налаштувань
```

Тепер розглянемо можливість використання методу часових затримок і отриманих у 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

xlabel = 'time, days' # підпис по вісі Ox
ylabel = symbol       # підпис по вісі Oy
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'  # підпис по вісі Ox
ylabel = symbol            # підпис по вісі Oy
```

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 Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

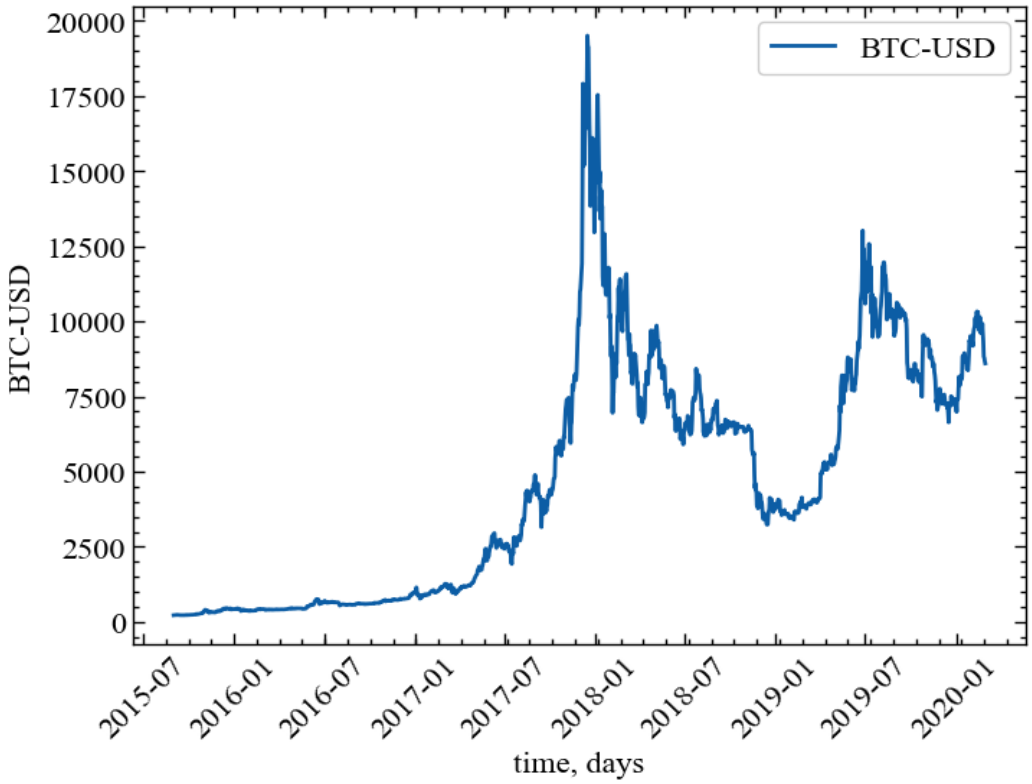


Рис. 2.5: Динаміка щоденних змін індексу Біткоїна

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 ε. У цьому випадку 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 ($d_E > 3$), ми послуговуватимемось значеннями $d_E = 2$ та $d_E = 3$. Значення τ 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");
```

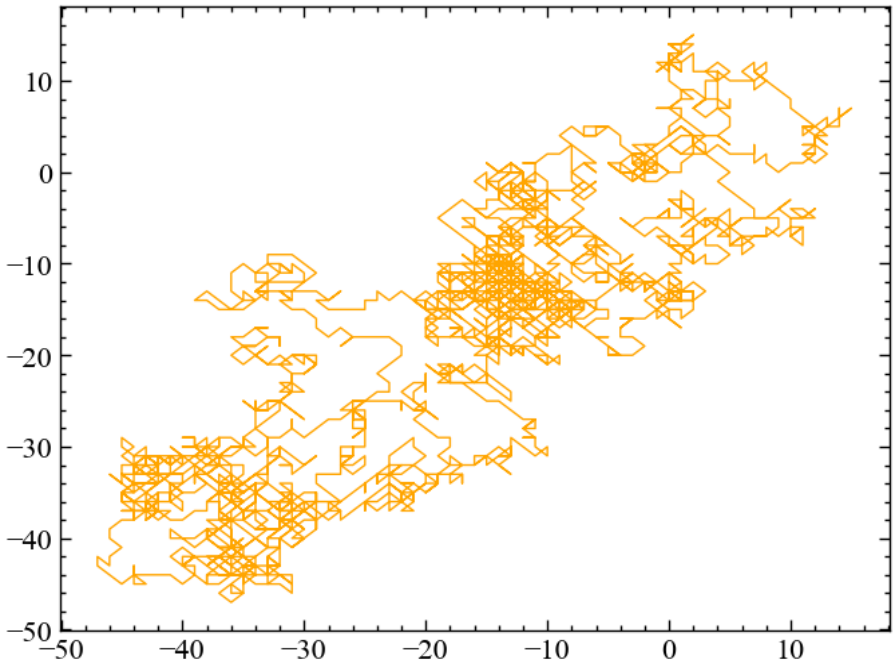


Рис. 2.6: Двовимірний фазовий портрет випадкового блукання

```
nk.complexity_attractor(nk.complexity_embedding(signal_random_walk, dimension=3,
delay=100), alpha=1, color="orange");
```

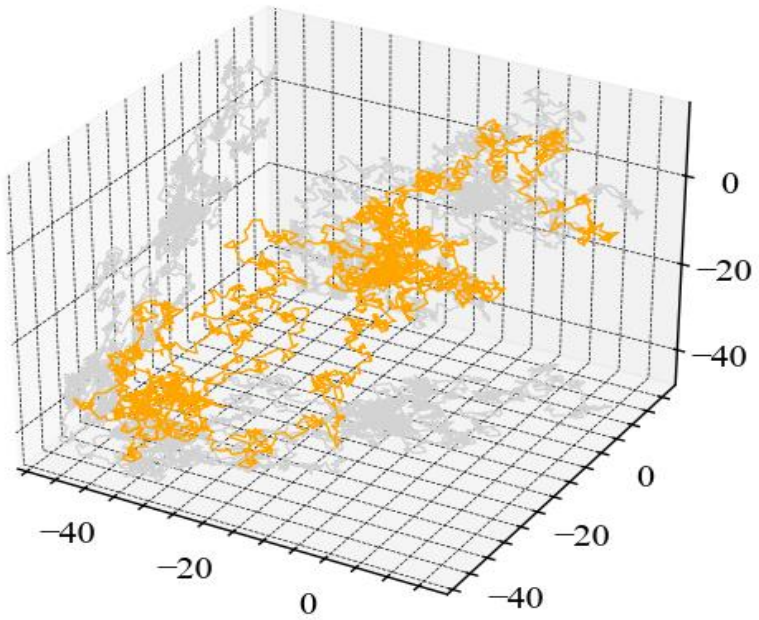


Рис. 2.7: Тривимірний фазовий портрет випадкового блукання

```
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");
```

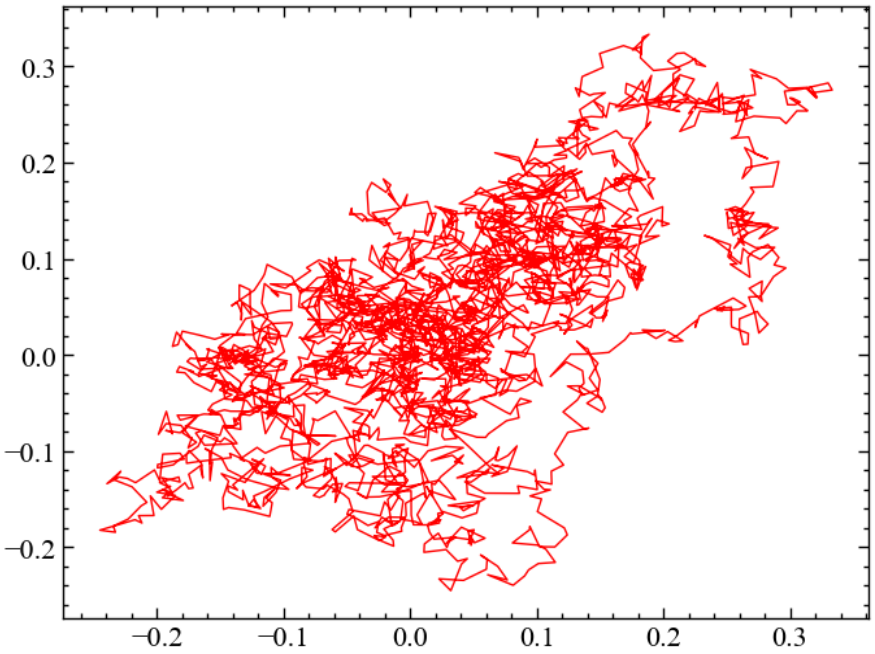


Рис. 2.8: Двовимірний фазовий портрет системи Орнштайна

```
nk.complexity_attractor(nk.complexity_embedding(signal_ornstein, dimension=3,
delay=100), alpha=1, color="red");
```

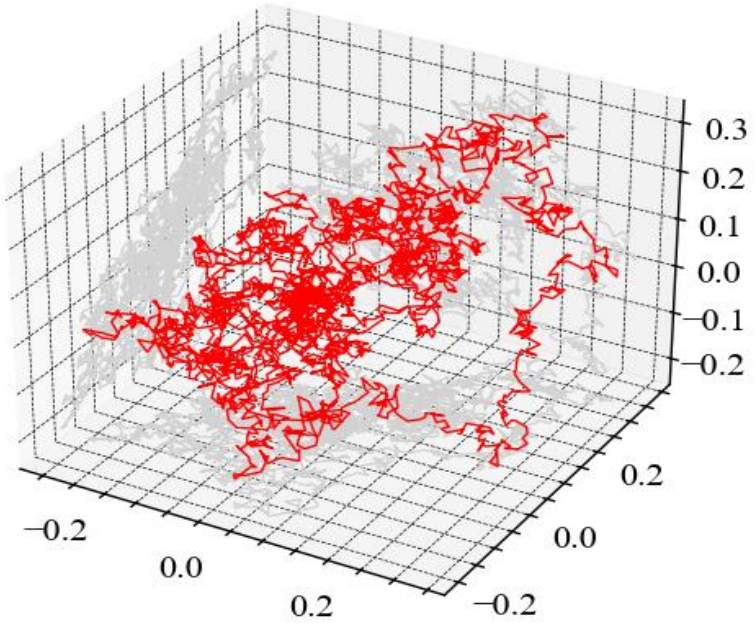


Рис. 2.9: Двовимірний фазовий портрет системи Орнштайна

```
nk.complexity_attractor(color ="last_dim", alpha="time", duration=1);
```

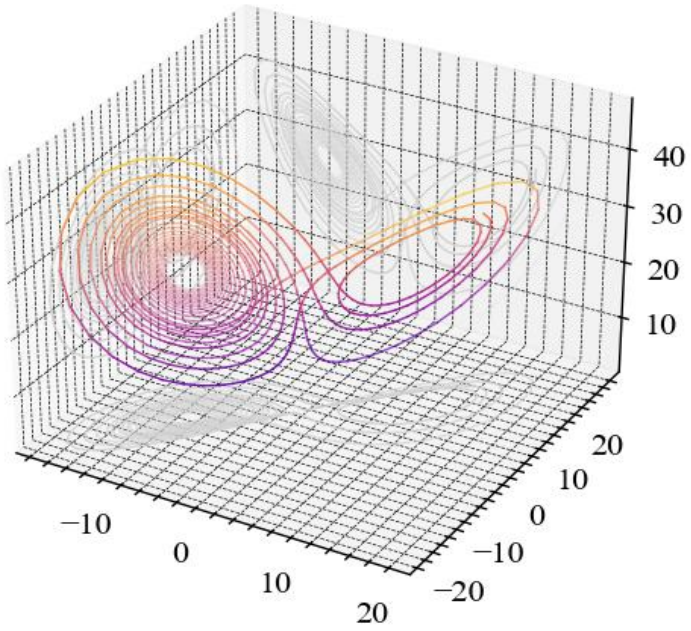


Рис. 2.10: Тривимірний фазовий портрет атрактора Лоренца

```
nk.complexity_attractor("rossler", color ="blue", alpha=1, sampling_rate=5000);
```

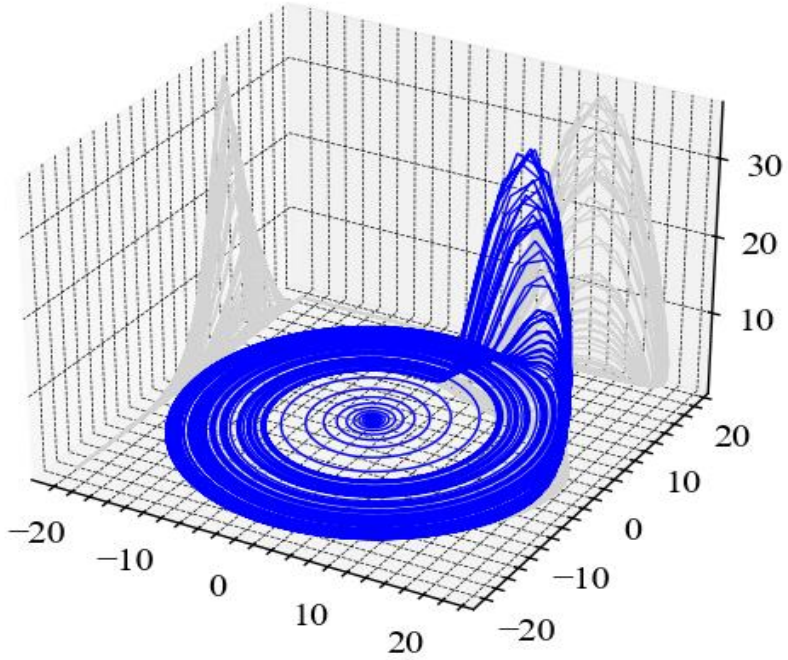


Рис. 2.11: Тривимірний фазовий портрет атрактора Рьосслера

```
nk.complexity_attractor(nk.complexity_embedding(for_rec, dimension=2,
delay=100), alpha=1, color="maroon");
```

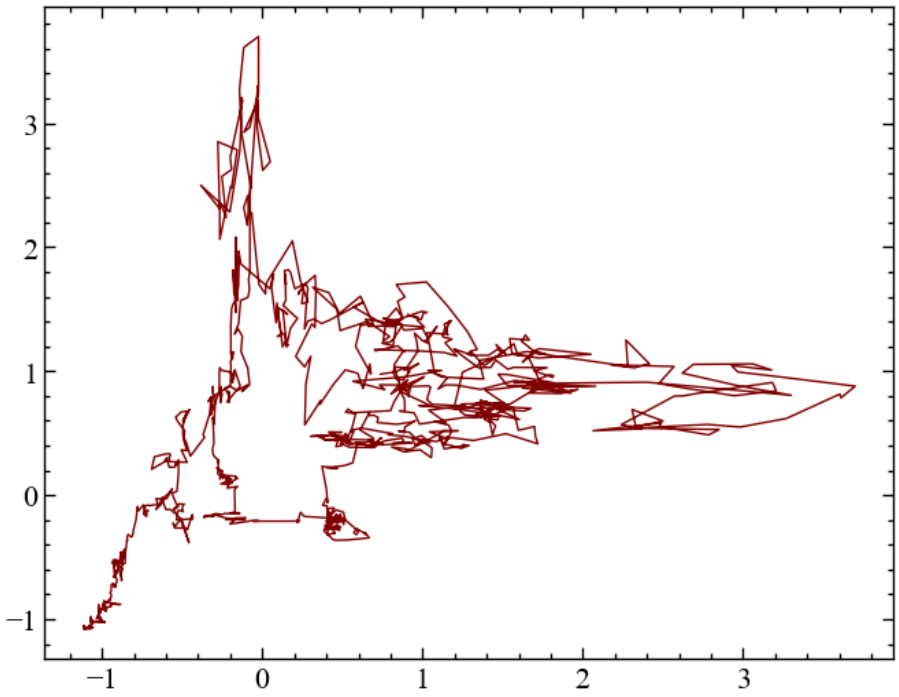


Рис. 2.12: Двовимірний фазовий портрет вихідних значень досліджуваного ряду Біткоїна

```
nk.complexity_attractor(nk.complexity_embedding(for_rec, dimension=3,
delay=100), alpha=1, color="maroon");
```

2.2.2 Побудова рекурентної матриці

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

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 у вигляді вектора значень;
- **delay** (*int*) — затримка в часі;
- **dimension** (*int*) — розмірність вкладень, d_E;
- **tolerance** (*float*) — радіус ε багатовимірного околу, в межах якого шукаються рекурентні траєкторії, а дві точки даних вважаються схожими. Якщо `"sd"` (за замовчуванням), буде встановлено значення $0.2 \cdot SD_{signal}$.

Емпіричним правилом є встановлення ε 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 фрагменту вихідних значень Біткоїна, його прибутковостей та стандартизованого фоагменту його вихідного ряду. Розмірність $d_E = 4$, часова затримка $\tau = 1$. Спробуємо різні варіанти ε:

- для вихідного ряду $\varepsilon = 100$;
- для прибутковостей та стандартизованого ряду $\varepsilon = 0.8$.

> ⓘ Примітка для побудови матриці рекурентності
>
> Як уже зазначалось на початку, рекурентна діаграма — це двовимірна квадратна двійкова матриця розміром N^2. Одним із її основних недоліків є те, що вона погано масштабується на великих даних. Наприклад, якщо ви плануєте досліджувати часовий ряд довжиною 10^6 значень, тоді рекурентна матриця буде складатись із 10^{12} білих і чорних точок, що може бути викликом для вашого процесора та оперативної пам'яті. Тому в подальшому ми пропонуємо будувати рекурентну матрицю не для всього ряду, а тільки для його підмножини. Для цього ми визначимо змінні початкового (`idx_beg`) та кінцевого (`inx_end`) відліку в межах якого буде здійснюватись побудова рекурентної матриці

```
idx_beg = 1000        # кінцевий відлік
idx_end = 1500        # початковий відлік

fragm = signal[idx_beg:idx_end]

# виконуємо приведення вихідного сигналу до прибутковостей
ret_type = 4
ret = transformation(fragm, ret_type)

# приводимо вихідний ряд до стандартизованого виду
ret_type = 6
for_rec = transformation(fragm, ret_type)

rc, _ = nk.recurrence_matrix(fragm,
                             delay=1,
                             dimension=4,
                             tolerance=100,
                             show=True)
```

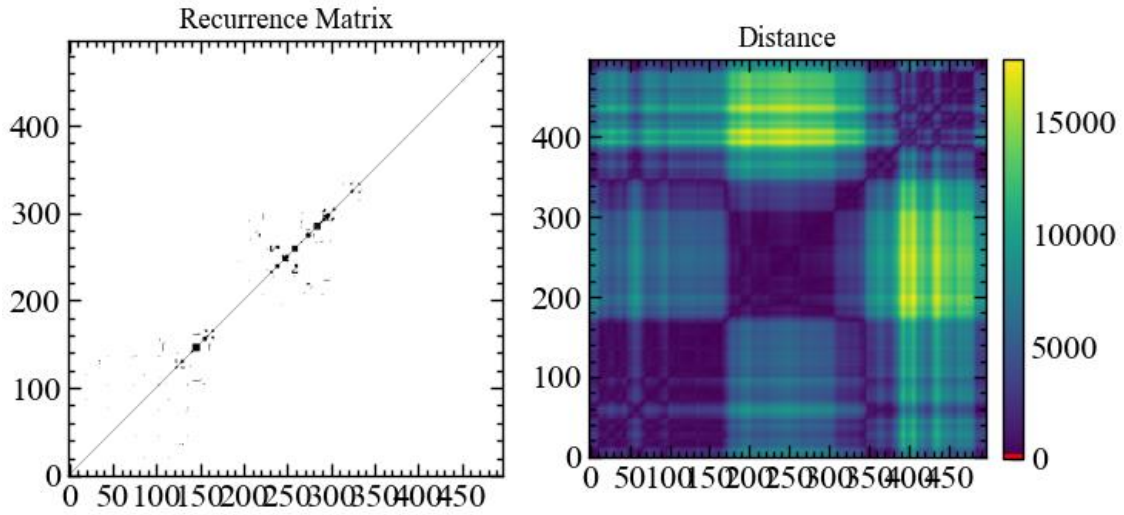


Рис. 2.14: Рекурентна матриця для вихідних значень Біткоїна

Як можна бачити з рисунку 2.14 всі траєкторії для простору вихідних значень за абсолютною шкалою залишаються доволі віддаленими один від одного. Для розпізнавання рекурентних закономірностей нам потребується поступово нарощувати ε. З цього рисунку видно, що $\varepsilon = 100$ буде замало.

Тепер спробуємо трохи нормалізувати значення вихідного фрагменту Біткоїна, аби реконструйовані траєкторії не знаходились занадто віддаленими один від одного. Для цього розрахуємо стандартизовані прибутковості:

```
# будуємо рекурентну матрицю
rc, _ = nk.recurrence_matrix(ret,
                             delay=1,
                             dimension=4,
                             tolerance=0.8,
                             show=True)
```

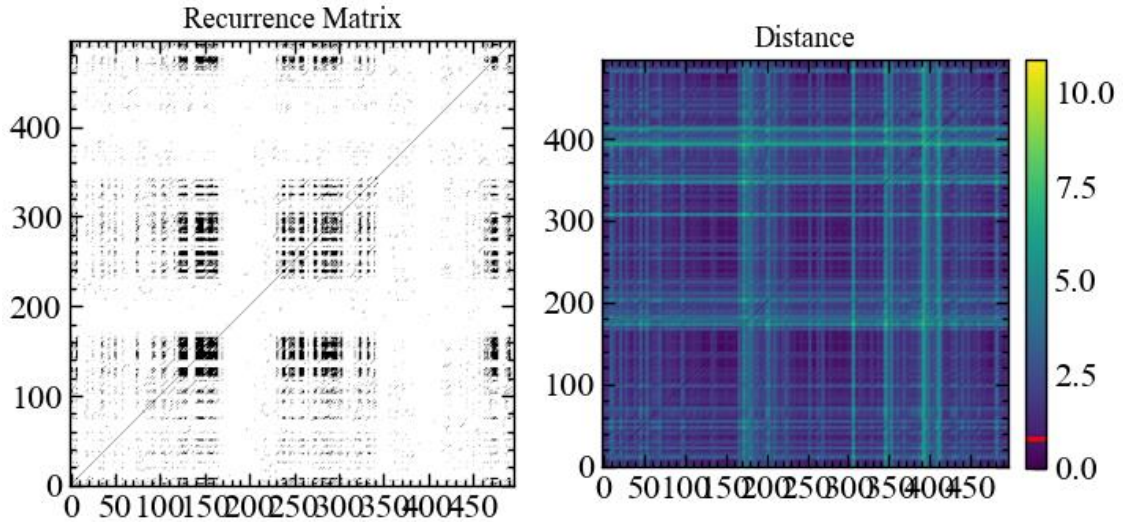


Рис. 2.15: Рекурентна матриця для стандартизованих прибутковостей Біткоїна

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:

```
# будуємо рекурентну матрицю
rc, _ = nk.recurrence_matrix(for_rec,
                             delay=1,
                             dimension=4,
                             tolerance=0.8,
                             show=True)
```

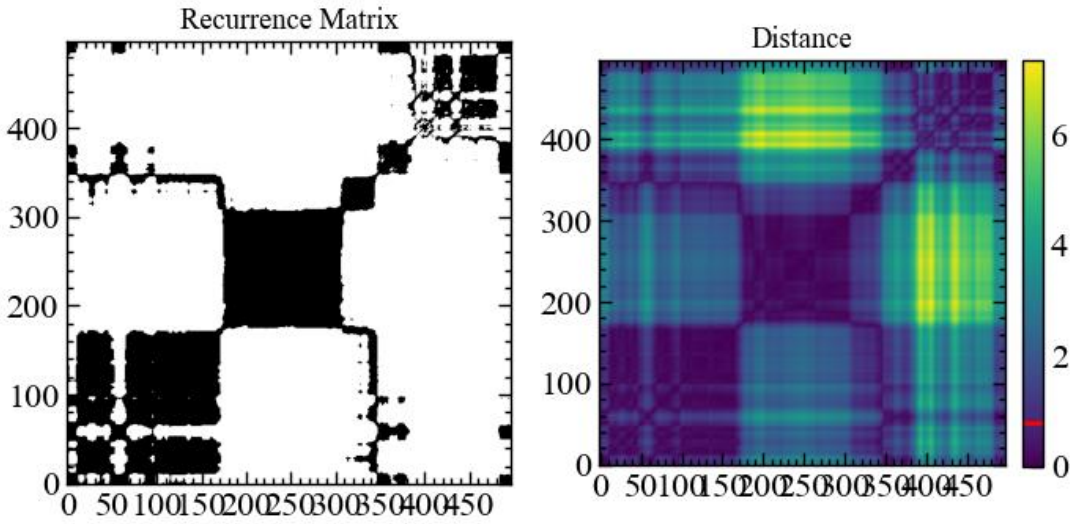


Рис. 2.16: Рекурентна матриця для стандартизованого вихідного ряду Біткоїна

На початку свого існування Біткоїн характеризувався доволі високим ступенем передбачуваності, низькою волатильністю коливань. Надалі почали домінувати білі області, а зараз Біткоїну властива динаміка схожа з броунівським рухом.

2.3 Висновок

У даній лабораторній роботі було продемонстровано можливість дослідження складних нелінійних систем із використанням рекурентних діаграм. На прикладі Біткоїна було показано, що складним фінансовим систем властива топологія узагальненого броунівського руху, де рекурентність може варіюватись

із плином часу. Тобто, той самий криптовалютний ринок характеризується мінливістю ступеня передбачуваності. Змога знаходження станів передбачуваності з точки зору рекурентних діаграм надає перспективи для їх використання разом із методами прогнозування та побудови індикаторів-передвісників.

2.4 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. Провести побудову фазових портретів і рекурентних діаграм для вашого часового ряду із використанням процедур автоматичного підбору параметрів розмірності вкладень і часової затримки. Порівняти результати з тими, що були отримані без використання даних методів і зробити висновки

> ⓘ Для виконання 4-го завдання самостійної роботи
>
> Далі надається опис алгоритмів для автоматизованого підбору розмірності реконструйованого фазового простору та часової затримки

2.4.1 Автоматизований підбір параметра часової затримки, τ

Часова затримка τ (також відома як L) є одним з двох критичних параметрів, що беруть участь у процедурі реконструкції фазового простору. Значення L 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 τ менше оптимального теоретичного значення, послідовні координати стану системи корельовані і атрактор недостатньо розгорнутий. І навпаки, коли τ 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)** [8] пропонують використовувати перший локальний мінімум взаємної інформації між затриманим і незатриманим часовими рядами, ефективно визначаючи значення τ, для якого вони діляться найменшою інформацією (і де атрактор є найменш надлишковим). На відміну від автокореляції, взаємна інформація враховує також нелінійні кореляції;
- **Тейлер (1990)** [9] запропонував вибирати таке значення τ, при якому автокореляція між сигналом та його зміщенною версією при τ вперше перетинає значення $1/\exp$. Методи, що базуються на автокореляції, мають перевагу за часом обчислень, коли вони знаходяться за допомогою алгоритму швидкого перетворення Фур'є (fast Fourier transform, FFT);
- **Касдаглі (1991)** [10] пропонує замість цього брати перший нульовий перетин автокореляції;
- **Розенштейн (1993, 1994)** [11,12] вважає, що слід апроксимувати точку, де функція автокореляцій падає до $(1 - 1/\exp)$ від свого максимального значення. Або ж наближатися до точки, близької до 40% нахилу середнього зміщення від діагоналі;
- **Кім (1999)** [13] оцінює τ 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 є значно довшим через необхідність порівнювати кожну унікальну пару парних векторів у реконструйованому сигналі на кожну затримку;
- **Лайл (2021)** [14] описує "Реконструкцію симетричного проекційного атрактора" (Symmetric Projection Attractor Reconstruction, SPAR), де $1/3$ 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. Цей метод також є найшвидшим, але може не підходити для аперіодичних сигналів. Аргумент `algorithm` (за замовчуванням `"fft"`) передається до аргументу `method` методу `signal_psd()`.

Можна також відмітити метод для об'єднаного підбору параметрів затримки та розмірності.

- **Гаутама (2003)** [15] зазначає, що на практиці часто використовують фіксовану часову затримку і відповідно регулюють розмірність вбудовування. Оскільки це може призвести до великих значень d_E (а отже, до вкладених даних великого розміру) і, відповідно, до повільної обробки, використовується метод оптимізації для спільного визначення d_E і τ на основі показника **entropy ratio**.

Розглянемо оптимальні значення розмірності та затримки для часового сигналу Біткоїна:

```
# виконуємо приведення вихідного сигналу до прибутковостей
ret_type = 4
ret = transformation(signal, ret_type)

# приводимо вихідний ряд до стандартизованого виду
ret_type = 6
for_rec = transformation(signal, ret_type)

delay, parameters = nk.complexity_delay(for_rec,
                                        delay_max=300, show=True,
                                        method="fraser1986")
```

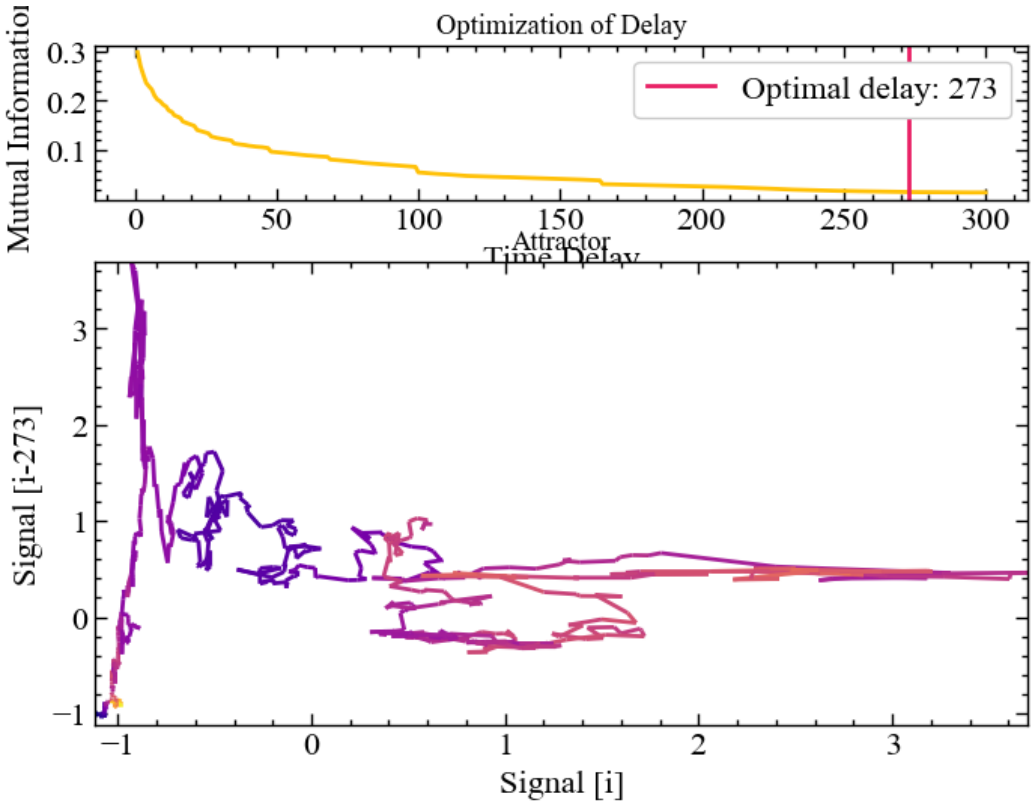


Рис. 2.17: Оптимальне значення часової затримки на основі методу Фрейзера і Свінні для часового ряду Біткоїна

Рис. 2.17 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")
```

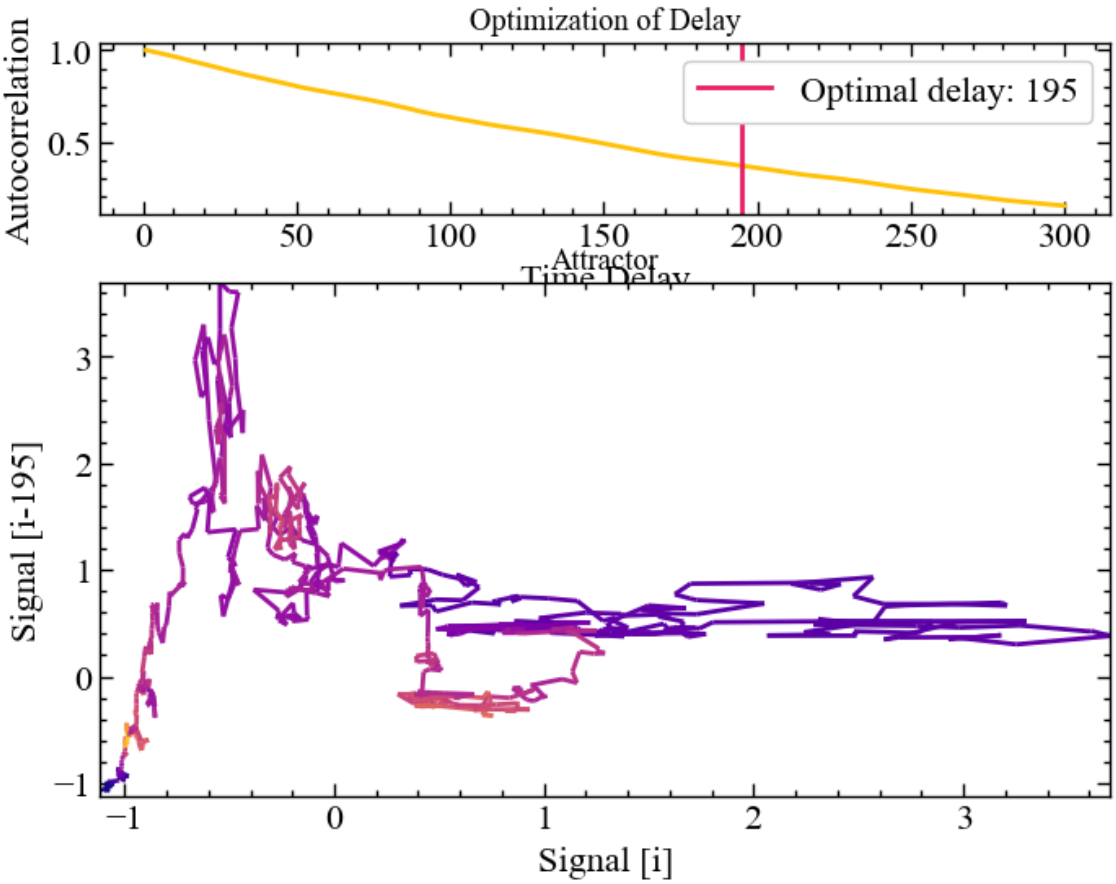


Рис. 2.18: Оптимальне значення часової затримки на основі методу Тейлера для часового ряду Біткоїна

Рис. 2.18 демонструє, що автокореляція між стандартизованих вихідним сигналом Біткоїна та його зміщенною версією при $\tau = 195$ вперше перетинає значення $1/\exp$. 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 $\tau = 195$ та $\tau = 273$ 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")
```

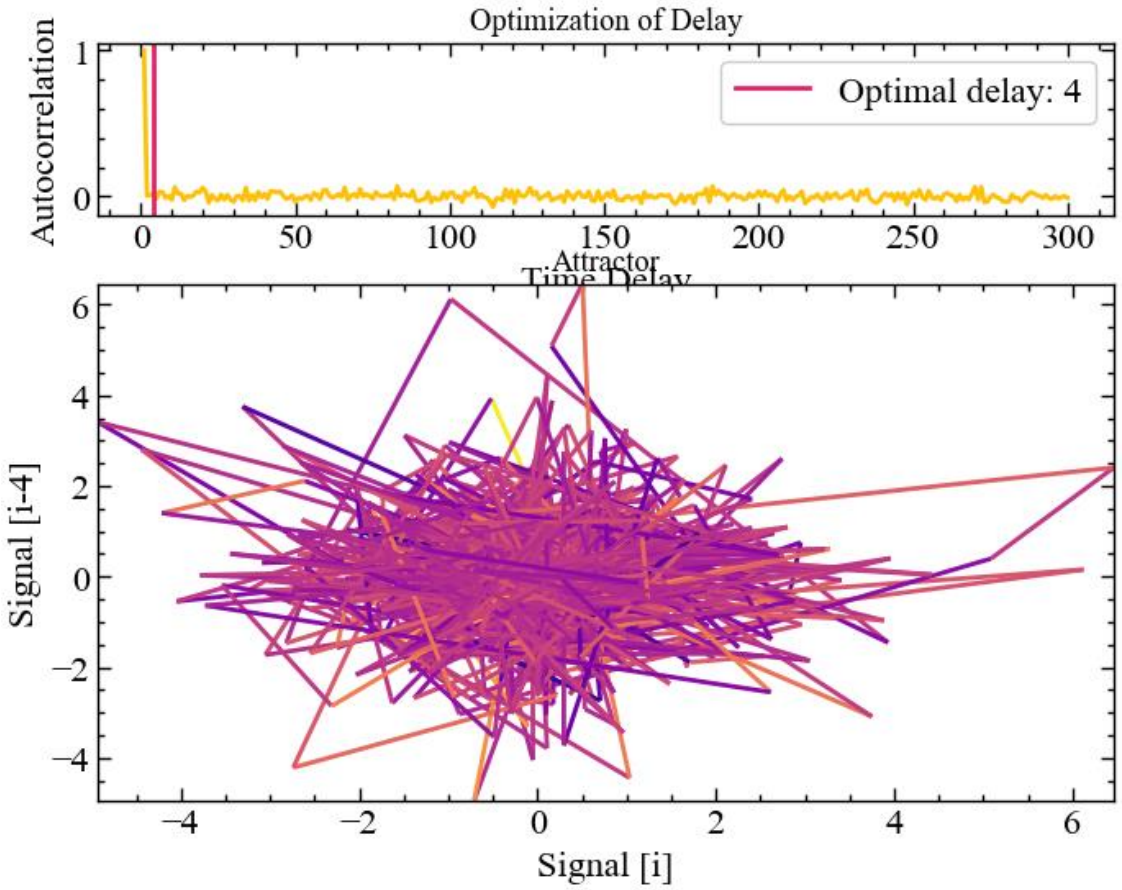


Рис. 2.19: Оптимальне значення часової затримки на основі методу Касдаглі для прибутковостей Біткоїна

Цього разу нам вдалося досягти оптимального результату, але приклад вище демонструє, що кожна процедура має свої виключення. Рис. 2.19 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")
```

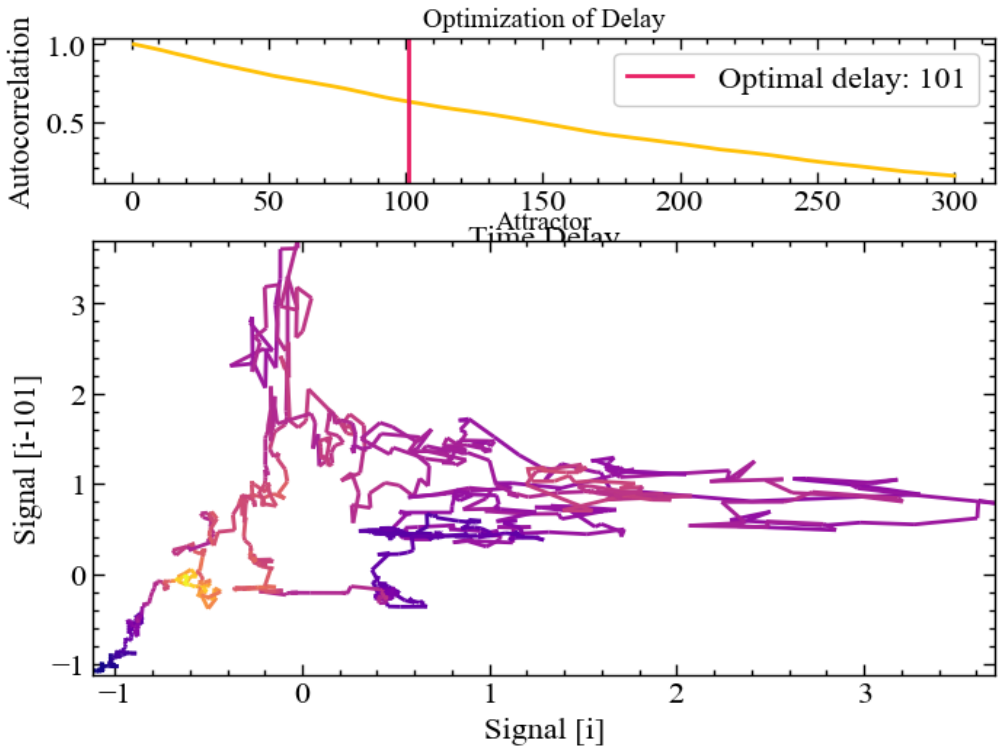


Рис. 2.20: Оптимальне значення часової затримки на основі методу Розенштайна (1993) для часового ряду Біткоїна

Рис. 2.20 демонструє, що при $\tau = 101$ функція автокореляцій перетинає значення $(1 - 1/\exp)$. 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")
```

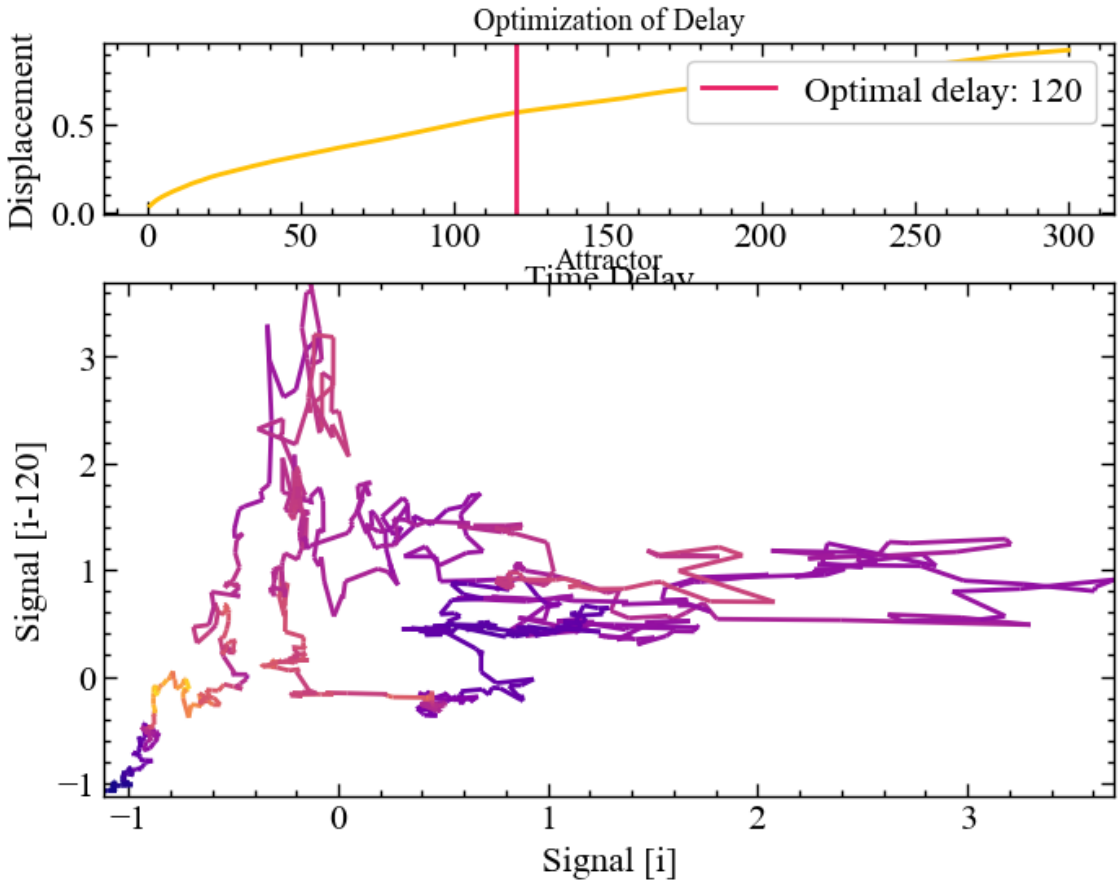


Рис. 2.21: Оптимальне значення часової затримки на основі методу Розенштайна (1994) для часового ряду Біткоїна

Рисунок вище показує, що при $\tau = 120$ 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")
```

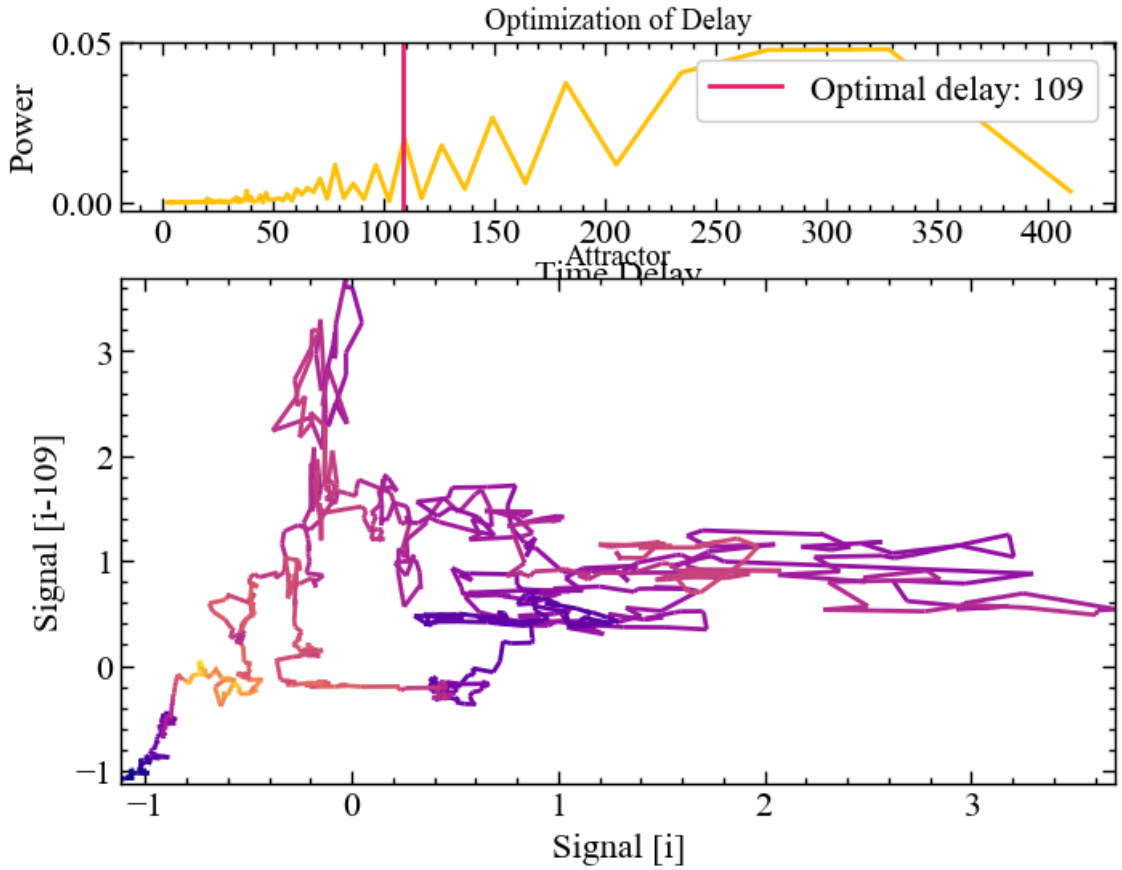


Рис. 2.22: Оптимальне значення часової затримки на основі методу Лайла для часового ряду Біткоїна

Згідно представленого вище результату найбільш значущі частоти, отримані за допомогою перетворення Фур'є, зберігаються при $\tau = 109$.

Тепер подивимось як це виглядатиме для об'єднаного підбору параметрів:

```
delay, parameters = nk.complexity_delay(for_rec,
    delay_max=np.arange(1, 10, 1), # діапазон значень затримки
    dimension_max=10,              # максимальна розмірність вкладень
    method="gautama2003",
    surrogate_n=5,                 # Кількість сурогатних сигналів
                                   # для генерації
    surrogate_method="random",     # Спосіб генерації сигналів
    show=True)
```

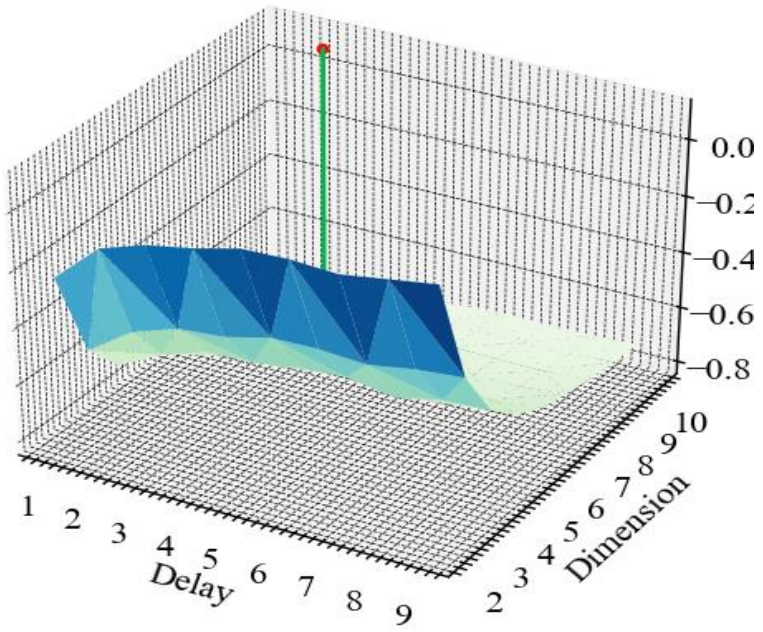


Рис. 2.23: Оптимальне значення розмірності та затримки на основі методу Гаутами для часового ряду Біткоїна

```
dimension = parameters["Dimension"]
dimension

10
```

Оскільки представлена вище процедура є доволі громіздкою в плані обчислювальних потужностей, ми обрали діапазон τ в межах від 1 до 10. Видно, що при τ близької до 3 оптимальне значення розмірності атрактора дорівнює 10. Можливо, при значеннях τ близьких до 100 або 200, ми могли б отримати зовсім інше значення розмірності, але це потребує додаткових експериментів.

2.4.2 Автоматизований підбір параметра розмірності вкладень, d_E

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 $d_E = 2$ або $d_E = 3$, але різні автори пропонують наступні процедури підбору:

- **кореляційна розмірність (Correlation Dimension, CD)**: Одним з перших методів оцінки оптимального d_E був розрахунок кореляційної розмірності для вкладень різного розміру і пошук насичення (тобто плато) в її значенні при збільшенні розміру векторів [16–18]. 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. [19–21], 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 d_E до $d_E + 1$. 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 d_E). На рисунку нижче (Рис. 2.24) 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;

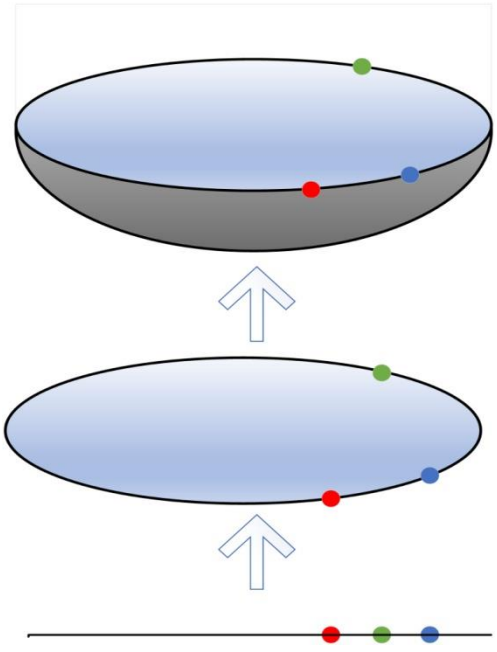


Рис. 2.24: Основна ідея методу FNN. Найближчі сусіди зеленої точки з’являються у випадку 1-, 2- та 3-вимірних фазових просторів [22]

- **середні хибні сусіди (Average False Neighbors, AFN)**: Ця модифікація методу FNN розроблена Cao (1997) [23] і усуває один з його основних

недоліків — необхідність евристичного вибору порогових значень r. Метод використовує максимальну евклідову відстань для представлення найближчих сусідів і усереднює всі відношення відстані в $d_E + 1$ розмірності до розмірності d_E та визначає *E1* та *E2* як параметри. Оптимальна розмірність досягається тоді, коли *E1* перестає змінюватися (виходить на плато). Це відбувається 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 d, вказує на випадковість даних, оскільки майбутні значення не залежать від минулих.

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 $R = 10.0$ або $A = 2.0$ (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 взяти $\tau = 100$. Приблизно таке значення спостерігалося для кожної процедури.

```
optimal_dimension, info = nk.complexity_dimension(for_rec,
                                                  delay=100,
                                                  dimension_max=10,
                                                  method='cd',
                                                  show=True)
```

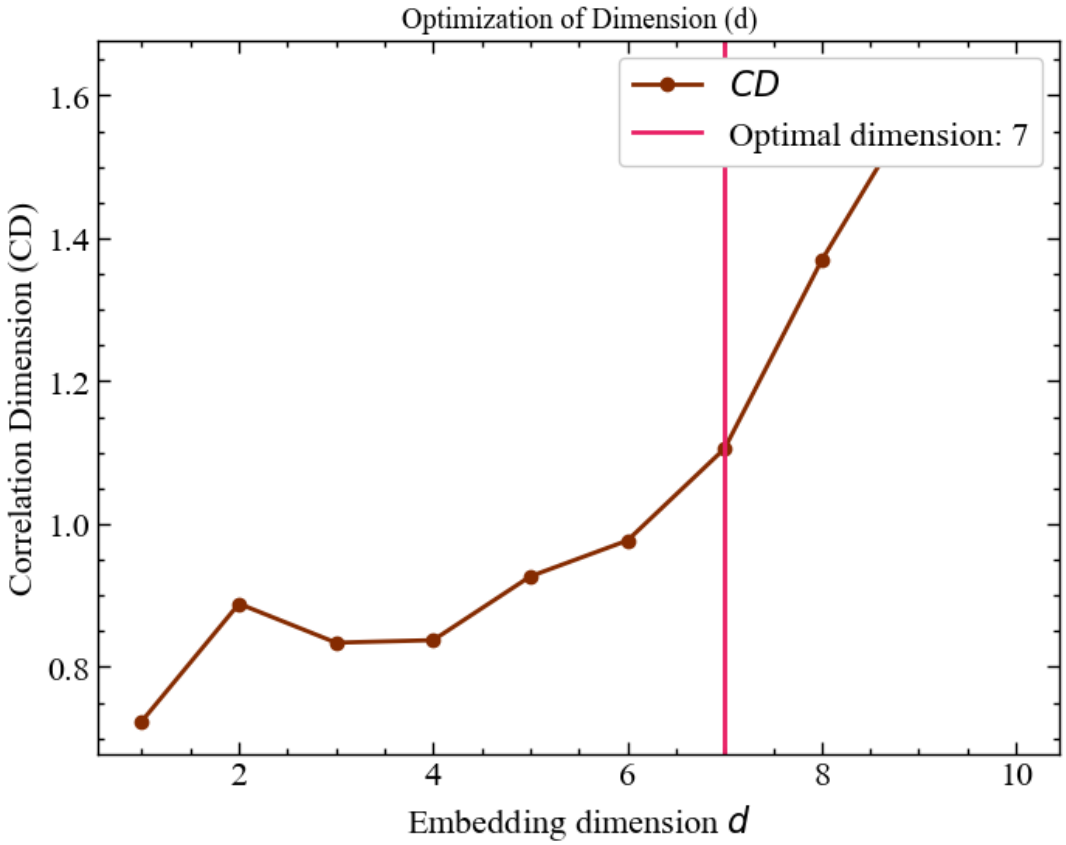


Рис. 2.25: Оптимальне значення розмірності на основі кореляційної розмірності для часового ряду Біткоїна

Рис. 2.25 демонструє той факт, що оптимальна розмірність вкладень, за якої 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)
```

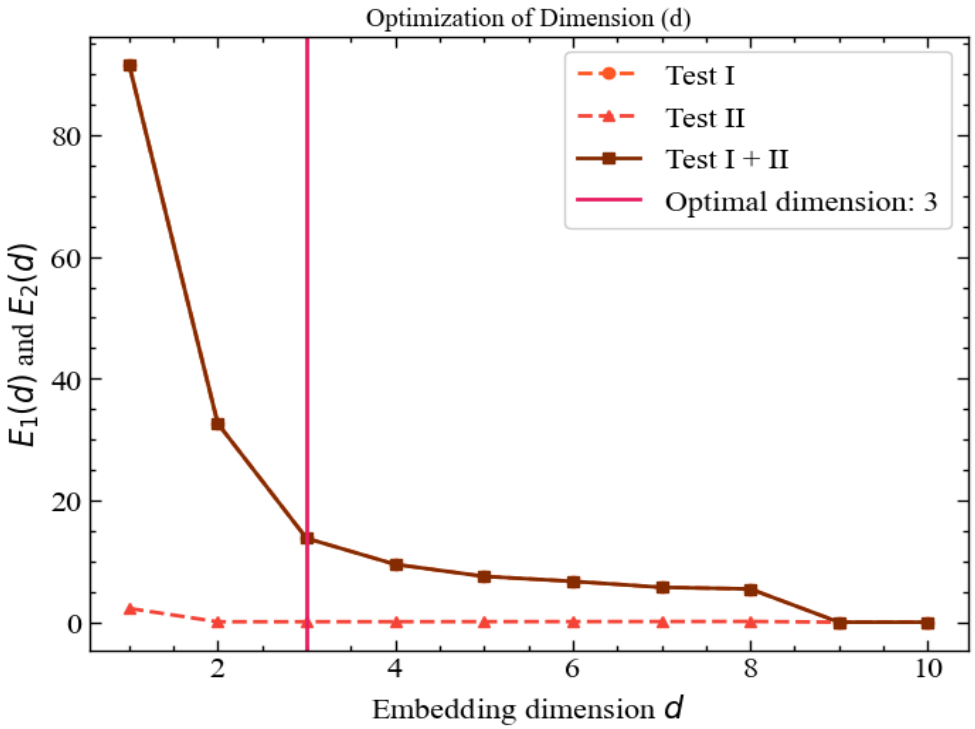


Рис. 2.26: Оптимальне значення розмірності на основі найближчих хибних сусідів для часового ряду Біткоїна

З рисунку 2.26 видно, що мінімальна розмірність вкладення 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 зростає.

```
optimal_dimension, info = nk.complexity_dimension(for_rec,
                                                   delay=20,
                                                   dimension_max=20,
                                                   method='afnn',
                                                   show=True)
```

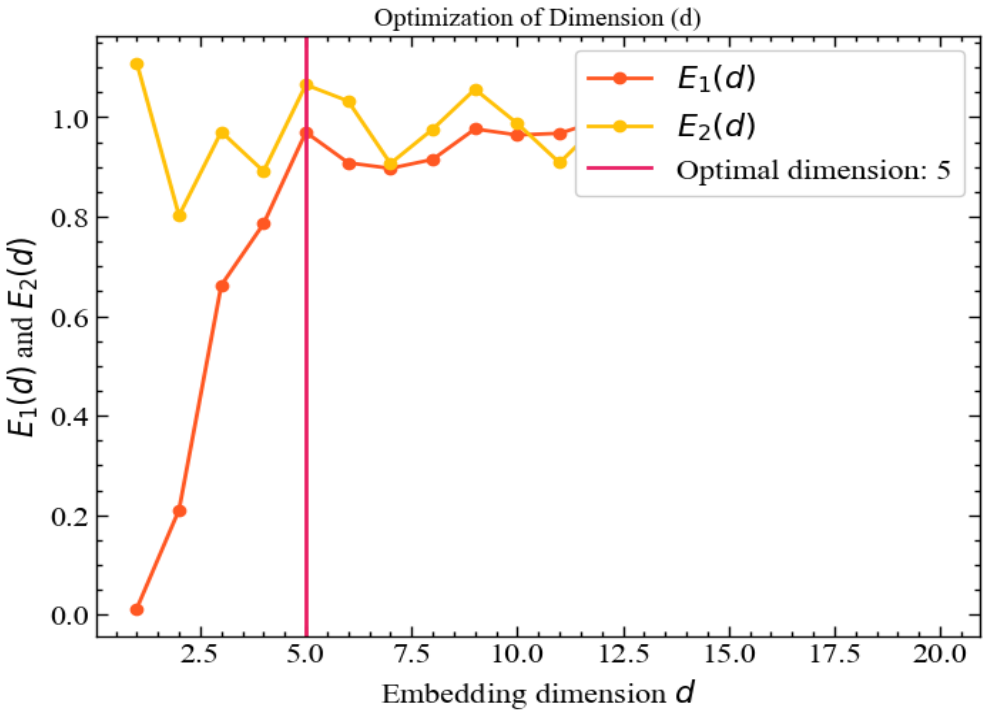


Рис. 2.27: Оптимальне значення розмірності на основі середніх найближчих хибних сусідів для часового ряду Біткоїна

Алгоритм середніх хибних сусідів показує, що тут розмірність вкладень $d_E = 5$ є оптимальною. При подальшому зростанні розмірності, атрактор стає більш стохастичним, що вказує на втрату всіх кореляцій.

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. Лабораторна робота № 3

Тема. Кількісний аналіз рекурентних діаграм

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

3.1 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 N^2 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 [24,25] були введені визначення та процедури для кількісної оцінки складності рекурентних діаграм, а згодом вони були розширені Марваном та ін [26].

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) [27].

3.2 Хід роботи

Перш ніж переходити до опису кожної з мір та їх розрахунків, визначимось з інструментарієм для виконання RQA. Як і до цього, ми використаємо бібліотеку `neuralkit2`.

Імпортуємо бібліотеки для подальшої роботи:

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

```
import neurokit2 as nk
import yfinance as yf
import scienceplots
import pandas as pd
from tqdm import tqdm

%matplotlib inline
```

І виконаємо налаштування рисунків для виводу:

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                   # розмір фонтів рисунку
'lines.linewidth': 2,                # товщина ліній
'axes.titlesize': 'small',           # розмір титулки над рисунком
'axes.labelsize': size,              # розмір підписів по осям
'legend.fontsize': size,             # розмір легенди
'xtick.labelsize': size,             # розмір розмітки по осі Ox
'ytick.labelsize': size,             # розмір розмітки по осі Ox
"font.family": "Serif",              # сімейство стилів підписів
"font.serif": ["Times New Roman"],   # стиль підпису
'savefig.dpi': 300,                  # якість збережених зображень
'axes.grid': False                   # побудова сітки на самому рисунку
}

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:

```
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 Ox
ylabel = symbol          # підпис по вісі Oy
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів і висновки залежать від того, з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'          # підпис по вісі Ox
ylabel = symbol                    # підпис по вісі Oy
```

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 Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

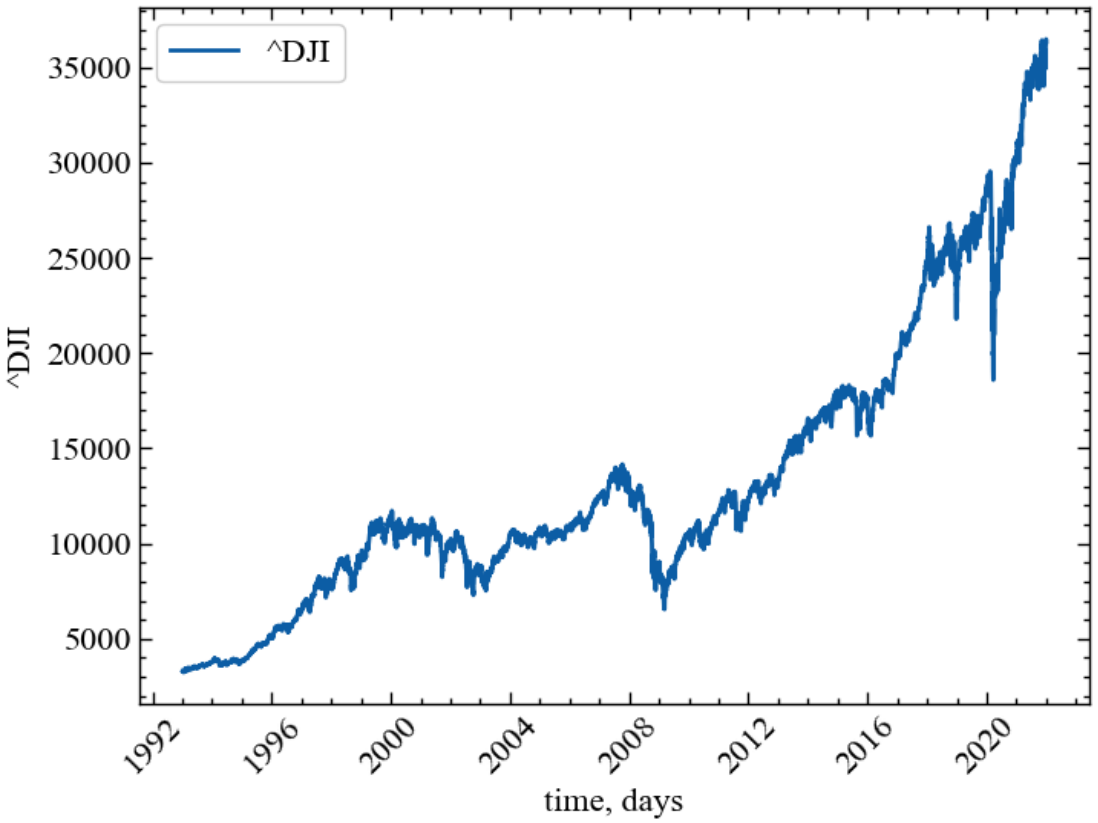


Рис. 3.1: Динаміка щоденних змін індексу Доу-Джонса

Користуючись тими методами, що ми розглянули в попередній лабораторній роботі, побудуємо атрактор даного ряду та його рекурентну діаграму. Але, перш за все, стандартизуємо 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:

> ⓘ Примітка для побудови матриці рекурентності
>
> Як уже зазначалось на початку, рекурентна діаграма — це двовимірна квадратна двійкова матриця розміром N^2. Тому в подальшому ми пропонуємо будувати рекурентну матрицю не для всього ряду, а тільки для його підмножини. Для цього ми визначимо змінні початкового (`idx_beg`) та кінцевого (`inx_end`) відліку в межах якого буде здійснюватись побудова рекурентної матриці

```
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

idx_beg = 3000 # кінцевий відлік
idx_end = 7300 # початковий відлік

fragm = signal[idx_beg:idx_end] # виокремлюємо фрагмент ряду

all_series = transformation(signal, ret_type) # виконуємо перетворення всього
ряду

for_rec = transformation(fragm, ret_type) # виконуємо перетворення фрагменту
```

На Рис. 3.2 представлено весь трансформований ряд та його фрагмент:

```
fig, ax = plt.subplots(1, 2)

ax[0].plot(all_series)
ax[0].set_title("Весь ряд")
ax[0].set_xlabel(xlabel)
ax[0].set_ylabel(ylabel)

ax[1].plot(for_rec)
ax[1].set_title("Фрагмент")
ax[1].set_xlabel(xlabel)

plt.show();
```

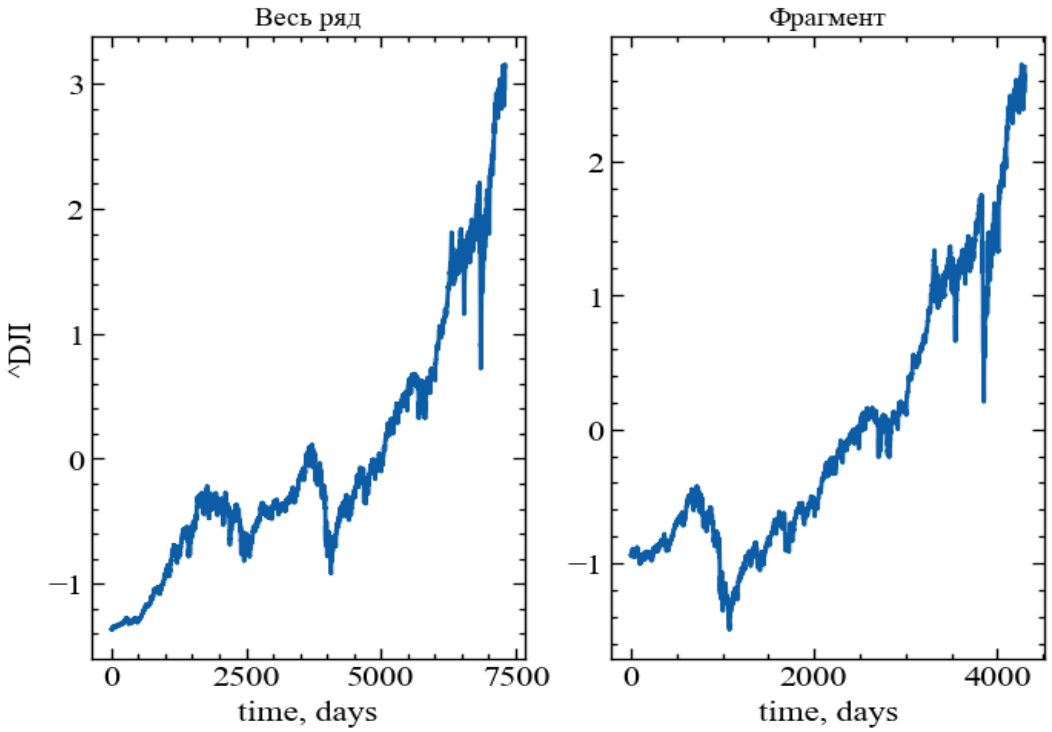


Рис. 3.2: Динаміка всього трансформованого ряду з використанням функції transformation() (рисунок зліва) та його фрагменту (рисунок справа)

Для всього ряду і для віконної процедури визначимо наступні параметри:

- розмірність вкладень $d_E = 3$;
- часова затримка $\tau = 1$;
- радіус багатовимірного околу $\varepsilon = 0.3$.

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");
```

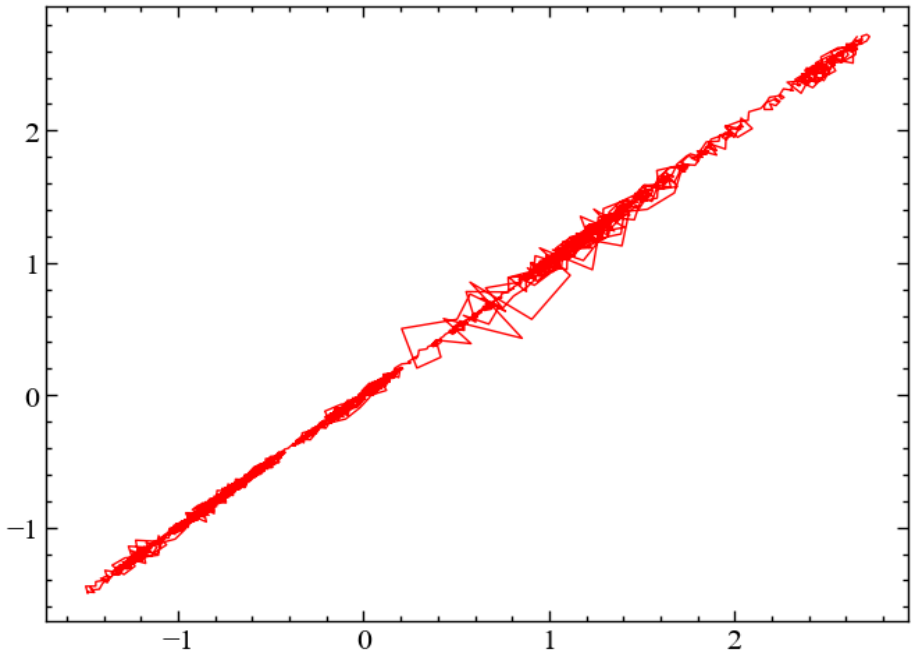


Рис. 3.3: Двовимірний фазовий портрет стандартизованих вихідних значень досліджуваного ряду Доу-Джонса

```
nk.complexity_attractor(nk.complexity_embedding(for_rec, dimension=3,
delay=tau), alpha=1, color="red");
```

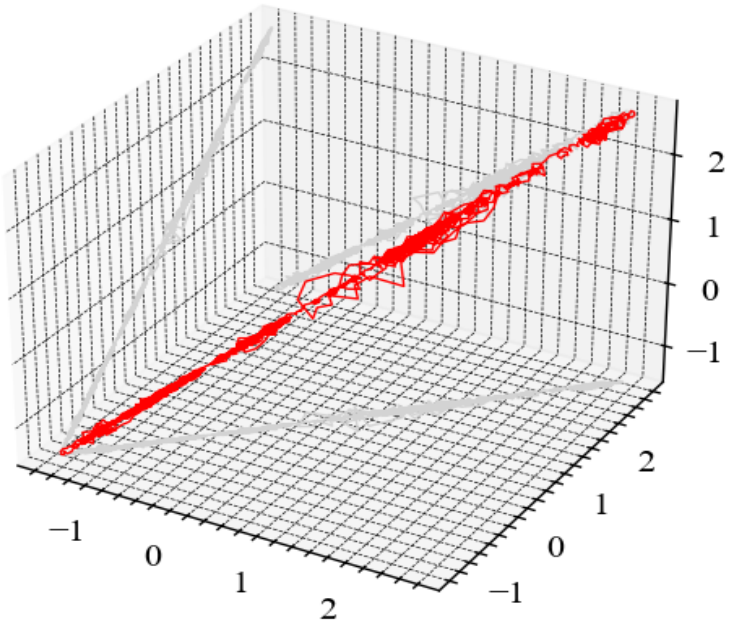


Рис. 3.4: Тривимірний фазовий портрет стандартизованих вихідних значень досліджуваного ряду Доу-Джонса

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)
```

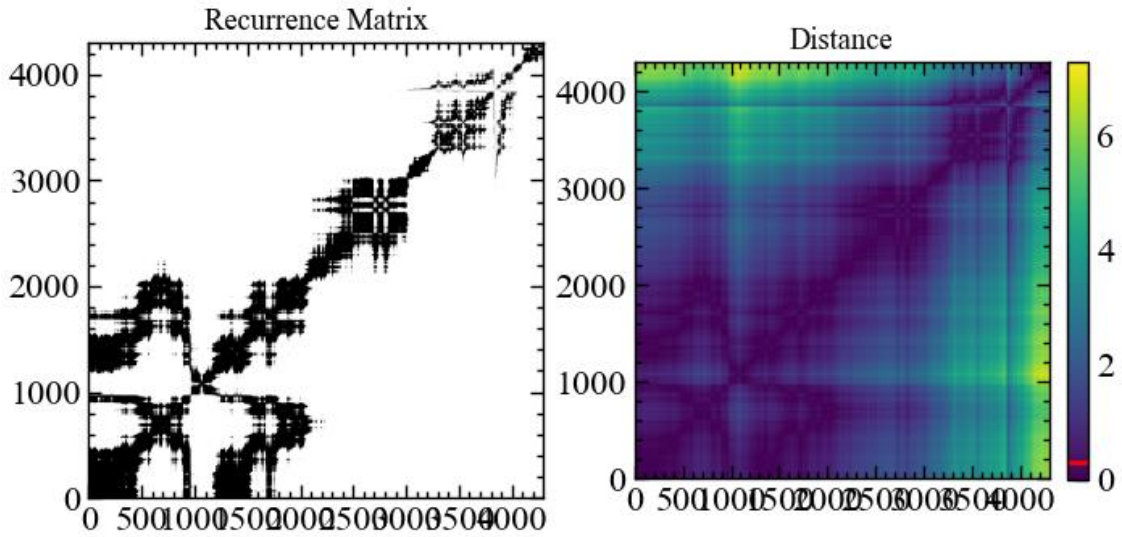


Рис. 3.5: Рекурентна матриця для стандартизованого вихідного ряду Доу-Джонса

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

Ми вже зазначали, що якісна репрезентація рекурентності станів не є достатньо об'єктивною. Найращим варіантом у даному випадку буде використання рекурентного аналізу за алгоритмом рухомого вікна.

3.2.1 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 використаємо метод `complexity_rqa()` бібліотеки `neuralkit2`. його синтаксис:

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, d_E;
- **tolerance** (*float*) — радіус ε багатовимірного околу в межах якого шукаються рекурентні траєкторії (часто позначається як r), відстань, на якій дві точки даних вважаються схожими. Якщо `"sd"` (за замовчуванням), буде встановлено значення $0.2 \cdot SD_{signal}$, де SD_{signal} визначає стандартне відхилення ряду;
- **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 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'])
```

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)
```

3.2.2 Рекурентні міри

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', rotation=45, **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();
```

3.2.2.1 Частота рекурентності

Найпростішим показником є **частота рекурентності** (recurrence rate), яка визначає щільність рекурентних точок на діаграмі, ігноруючи лінію ідентичності:

$$RR = \frac{1}{N^2} \sum_{i,j=1}^{N} R\,(i,j),$$

де N — кількість точок на траєкторії фазового простору.

Частота рекурентності відповідає ймовірності того, що певний стан повториться.

```
plot_recurrence_measure(measure=RR, label='RR')
```

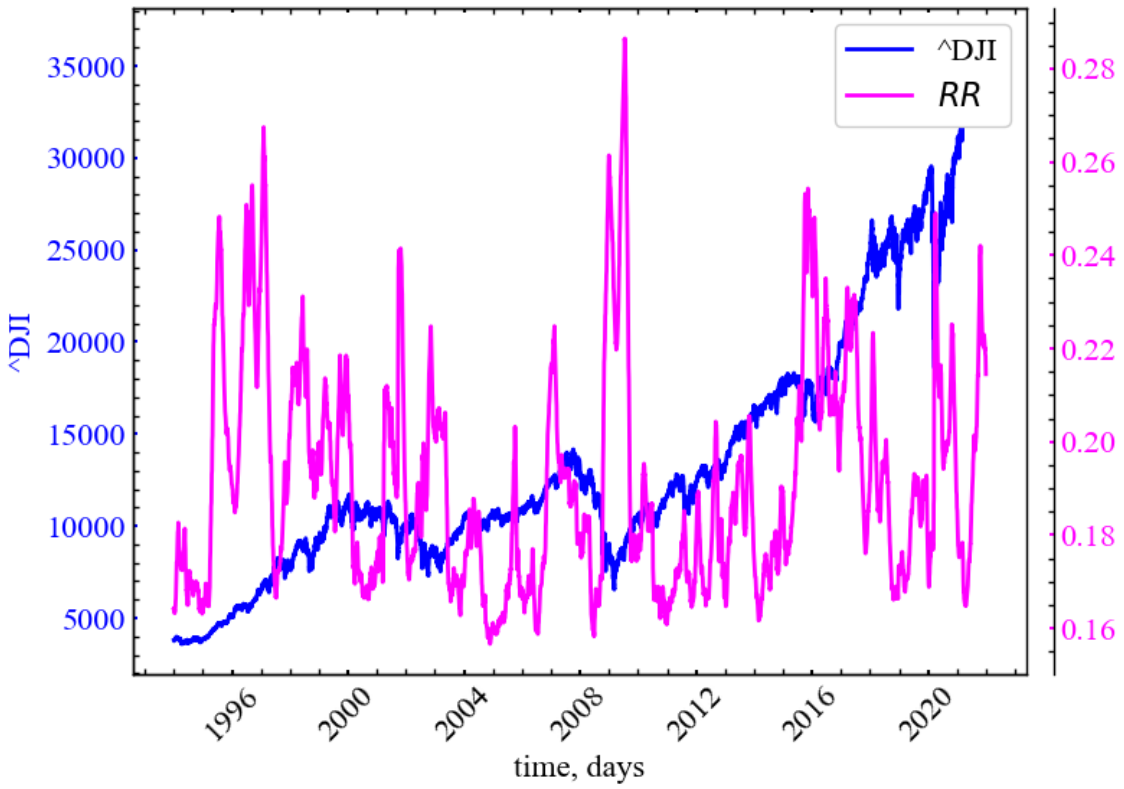


Рис. 3.6: Динаміка індексу Доу-Джонса та частоти рекурентності

Як ми можемо бачити з представленого рисунку (Рис. 3.6), міра рекурентності зростає при крахових подіях, що вказує на зростання ступеня самоорганізації та злагодженості торгівельної активності трейдерів на цьому ринку.

3.2.2.2 Діагональна частота рекурентності

Даний підхід базується на діагональних рекурентних профілях часового ряду [28]. Діагональний рекурентний профіль визначає кількість рекурентних точок на різних лагах подібно до функції автокореляцій. Для отримання діагонального профілю рекурентностей просто підраховується частка рекурентних точок на діагоналях, розташованих у нижньому правому або нижньому лівому куті діаграми, і будується графік як функція відстані від головної діагоналі, тобто лагу.

Іншими словами, **діагональна частота рекурентності** (diagonal recurrence rate) фіксує величину автокореляції на різних лагах.

```
plot_recurrence_measure(measure=DIAG_RR, label='DRR')
```

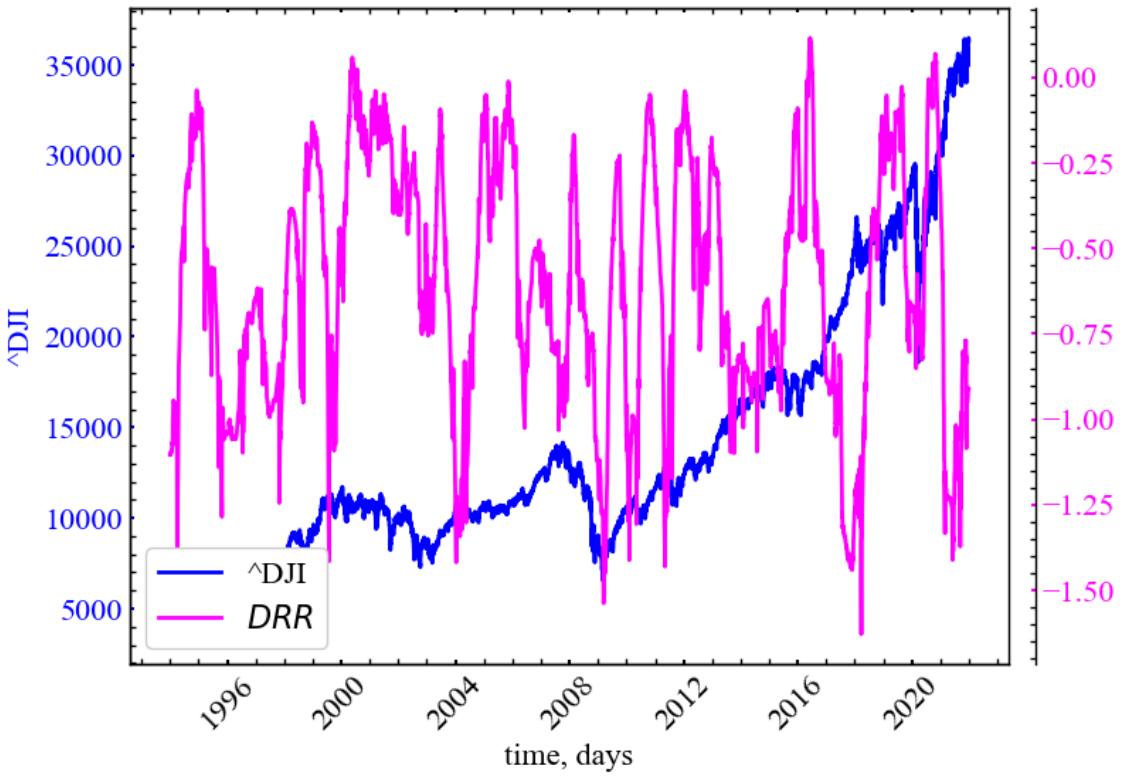


Рис. 3.7: Динаміка індексу Доу-Джонса та діагональної частоти рекурентності

З Рис. 3.7 видно, що діагональна частота рекурентності зростає у передкризові та кризові періоди, вказує на зростання величини автокореляції, що в свою чергу демонструє ріст ступеня самоорганізації.

3.2.2.3 Детермінізм

Наступним показником можна визначити частку рекурентних траєкторій, які формують діагональні лінії мінімальної довжини $\ell_{\min}$. Ця міра називається **детермінізмом** (determinism) і пов'язана з передбачуваністю динамічної системи:

$$DET = \sum_{\ell=\ell_{min}}^{N} \ell \cdot P(\ell) \Big/ \sum_{\ell=1}^{N} \ell \cdot P(\ell),$$

де $P(\ell)$ — частотний розподіл довжин ℓ діагональних ліній.

> Додаткова інформація по детермінізму

Детерміновані системи характеризуються значною варіацією діагональних ліній різної довжини. Періодичні сигнали характеризуються 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 закономірностей, що утворюватимуть дуже короткі діагональні лінії.

Білий шум, наприклад, мав би рекурентну діаграму з майже ізольованими рекурентними точками та дуже малим відсотком діагональних ліній, тоді як детермінований процес демонстрував би дуже малу кількість поодиноких рекурентностей, але велику щільність довгих діагональних ліній.

```
plot_recurrence_measure(measure=DET, label='DET')
```

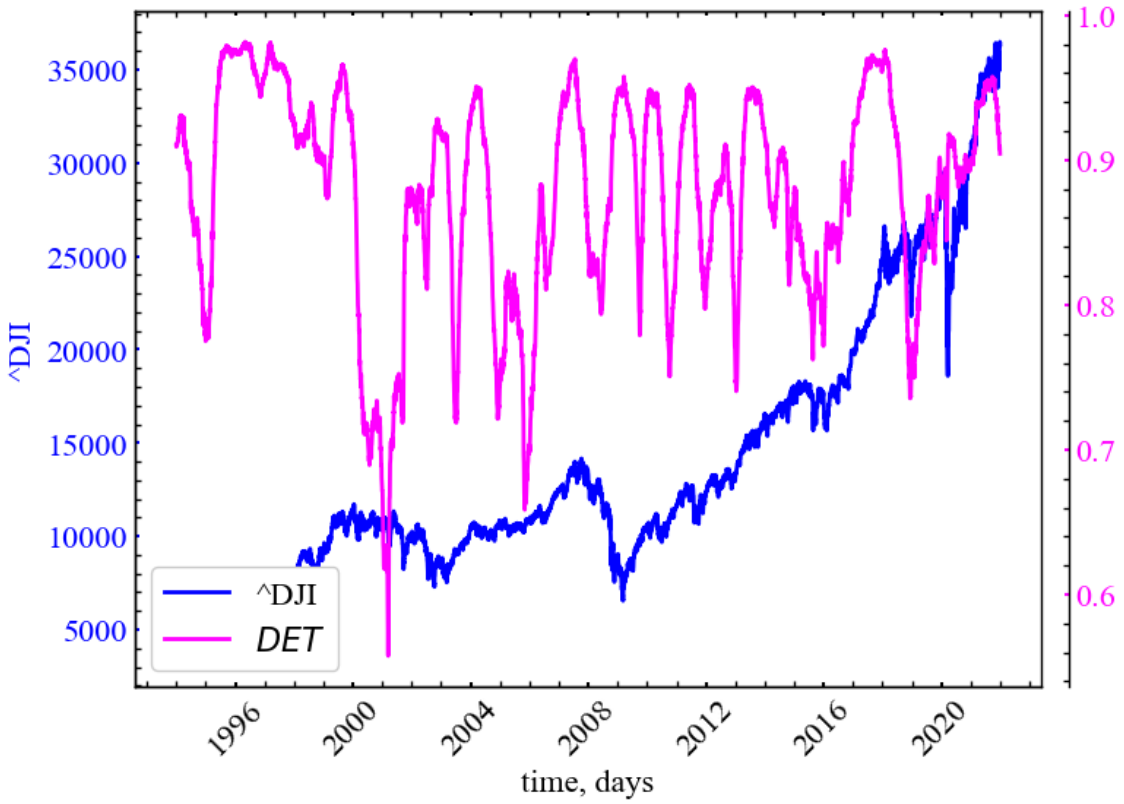


Рис. 3.8: Динаміка індексу Доу-Джонса та міри DET

Як ми можемо бачити з Рис. 3.8, у передкризові та кризові періоди показник детермінізму починає зростати, що свідчить і про зростання ступеня передбачуваності (впорядкованості) флуктуацій системи.

3.2.2.4 Ламінарність

Показник, що характеризує кількість рекурентних станів, які утворюють вертикальні лінії, називається **ламінарністю** (laminarity) і пов'язаний з кількістю ламінарних фаз (незмінностей) у системі:

$$LAM = \sum_{v=v_{min}}^{N} v \cdot P(v) \Big/ \sum_{v=1}^{N} v \cdot P(v),$$

а $P(v)$ — частотний розподіл довжин v вертикальних ліній, які мають довжину принаймні $v_{\min}$.

> Додаткова інформація по ламінарності
>
> Ламінарність характеризує ймовірність системи перебувати незмінному стані. Зі збільшенням ізольованих рекурентних точок у системі міра ламінарності спадатиме

```
plot_recurrence_measure(measure=LAM, label='LAM')
```

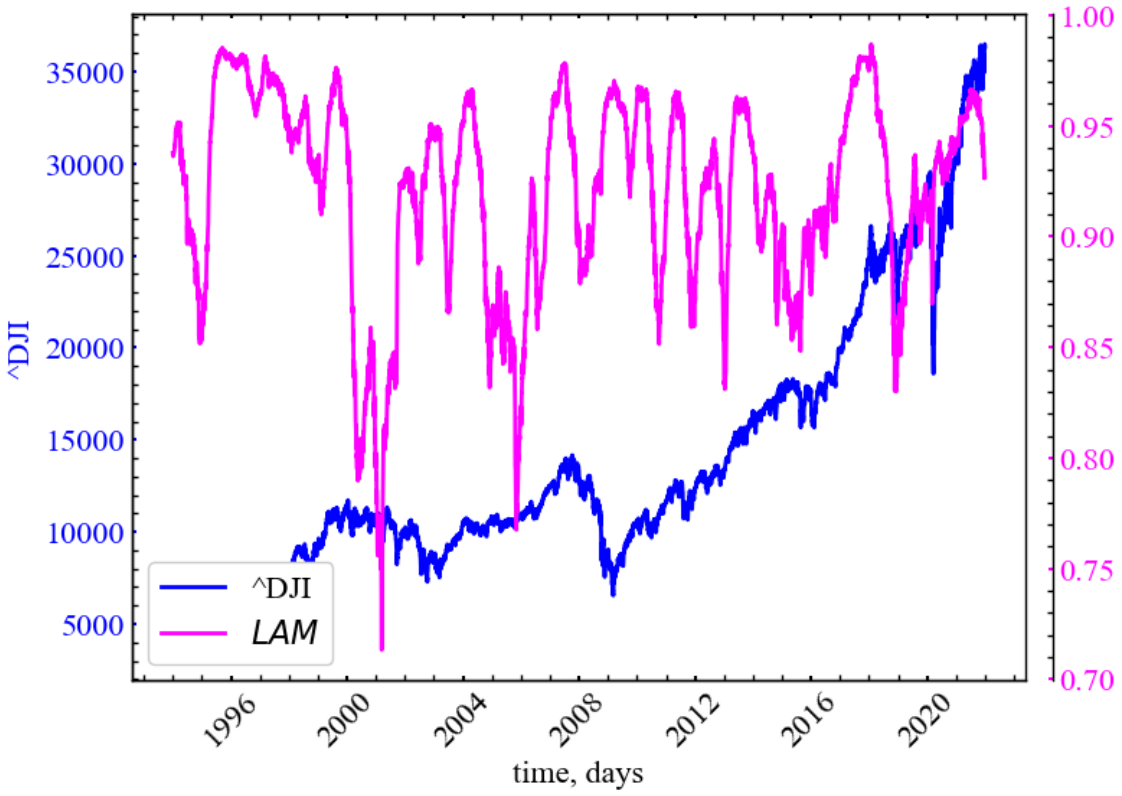


Рис. 3.9: Динаміка індексу Доу-Джонса та ламінарності

Можна бачити, що в умовах криз ступінь ламінарності зростає. Зростає і щільність діагональних точок, і загалом кількість рекурентних траєкторій у фазовому просторі. Кризи характеризуються трендостійкістю, персистентністю та детермінованістю своєї поведінки.

3.2.2.5 Середня довжина діагональних ліній

Також можна виміряти **середню довжину діагональних ліній** (average diagonal lines length). Середня довжина діагональних лінії визначається як

$$L = \sum_{\ell=\ell_{min}}^{N} \ell \cdot P(\ell) \Big/ \sum_{\ell=\ell_{min}}^{N} P(\ell).$$

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')
```

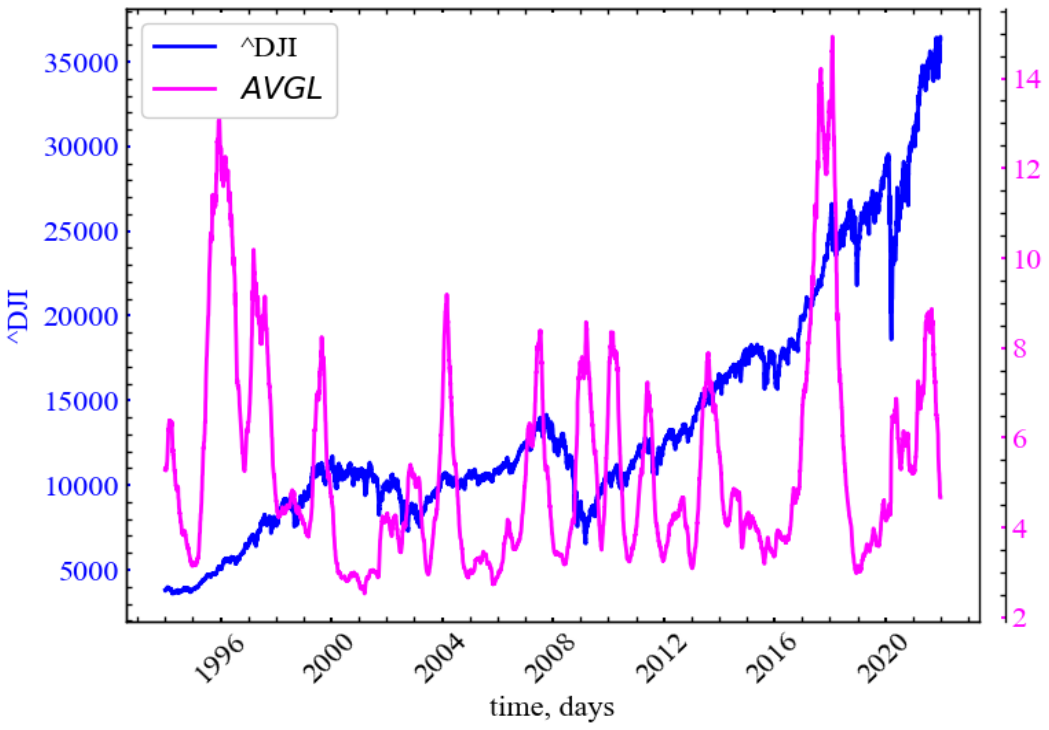


Рис. 3.10: Динаміка індексу Доу-Джонса та середньої довжини діагональних

ліній

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

3.2.2.6 Час захоплення/затримки

Усереднена довжина діагональної лінії пов'язана з часом передбачуваності динамічної системи та **часом затримки** (trapping time):

$$TT = \sum_{v=v_{min}}^{N} v \cdot P(v) \Big/ \sum_{v=v_{min}}^{N} P(v).$$

> 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 цієї величини характеризує дедалі більший час затримки досліджуваної системи в певному стані

```
plot_recurrence_measure(measure=TT, label='TT')
```

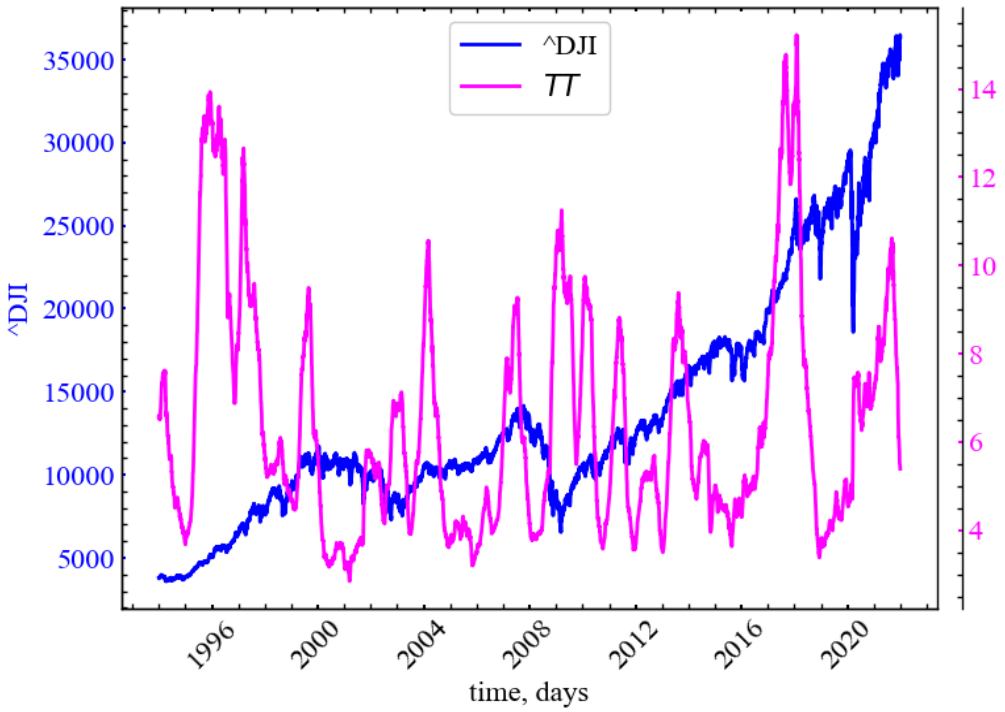


Рис. 3.11: Динаміка індексу Доу-Джонса та час затримки

На представленому рисунку (Рис. 3.11) видно, що TT зростає в (перед-)кризові стани, що вказує на намагання системи перебувати ще деякий деякий час у стані кризи.

3.2.2.7 Середня довжина білих вертикальних лінії

Середня довжина білих вертикальних ліній (average white vertical lines length) може бути визначена як

$$WVL_{mean} = \sum_{w=w_{min}}^{N} w \cdot P(w) \Big/ \sum_{w=w_{min}}^{N} P\,(w).$$

$P(w)$ — це частотний розподіл білих вертикальних ліній довжиною w, а w_{min} 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}')
```

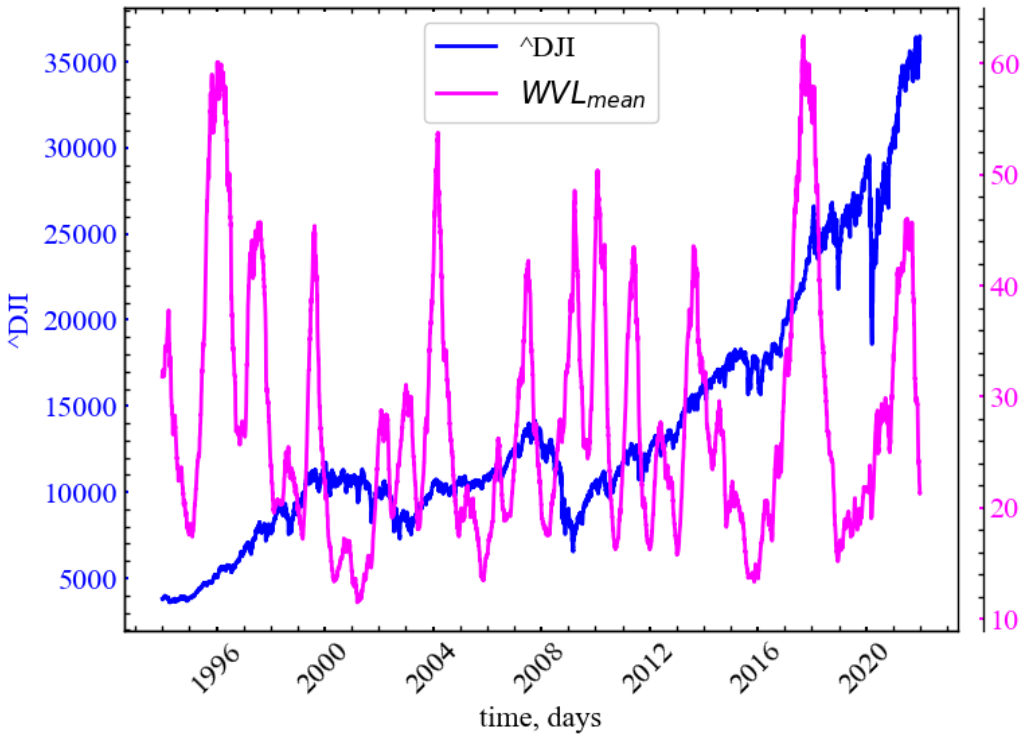


Рис. 3.12: Динаміка індексу Доу-Джонса та середньої довжини білих вертикальних ліній

Зростання середньої довжини білих вертикальних ліній на Рис. 3.12 демонструє, що кризові події характеризуються не лише детермінізмом динаміки фондового ринку, але й несхожістю даних подій у порівнянні з попередніми станами.

3.2.2.8 Ентропія діагональних ліній

Для відповідних діагональних сегментів можна розрахувати необхідну кількість інформації для опису всього розподілу цього типу ліній. Імовірність $p(\ell)$ того, що діагональна лінія має довжину ℓ, можна оцінити за частотним розподілом $P(\ell)$ із $p(\ell) = P(\ell)/\sum_{\ell=\ell_{\min}}^{N} P(\ell)$. **Ентропію Шеннона** ймовірності появи таких діагональних ліній (diagonal lines entropy) можна визначити наступним чином:

$$DLEn = -\sum_{\ell=\ell_{min}}^{N} p(\ell)\ln p(\ell).$$

Даний показник відображає складність досліджуваної структури.

Додаткова інформація по ентропії діагональних ліній

Для некорельованого шуму чи осциляцій ми тримали б мале значення ентропії, що вказувало б на асиметричний розподіл діагональних ліній: існувала б невеличка частка діагональних ліній конкретної довжини, що характеризувала б рекурентність досліджуваної системи. Зростання даної ентропії свідчило б про зростання симетричності розподілу довжин діагональних ліній

```
plot_recurrence_measure(measure=ENT_DIAG, label='DLEn')
```

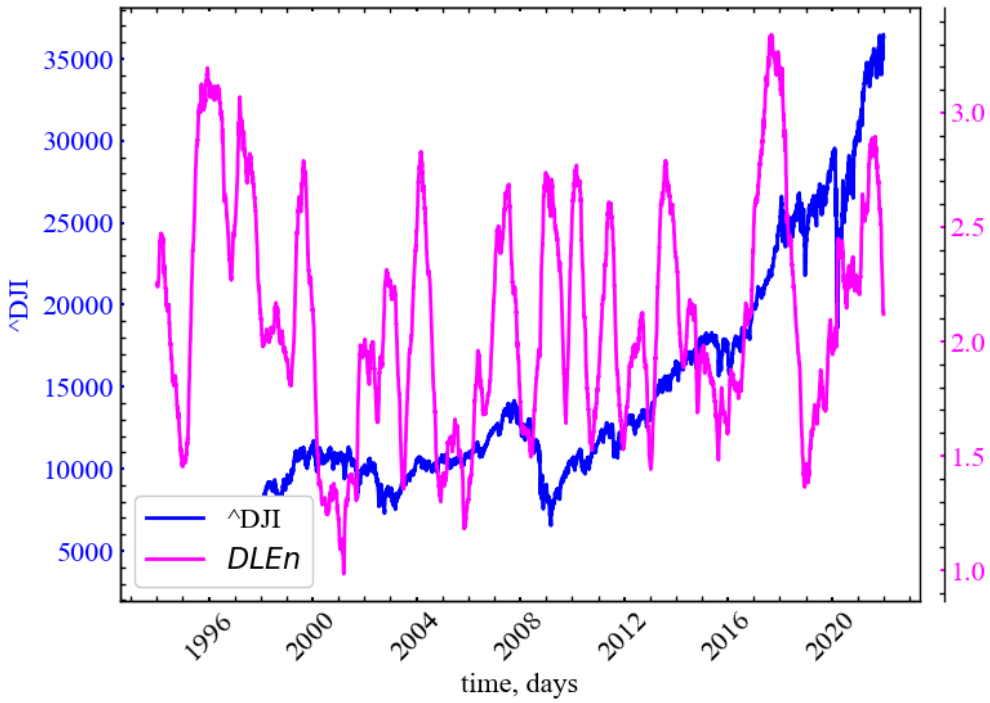


Рис. 3.13: Динаміка індексу Доу-Джонса та ентропії діагональних ліній

На Рис. 3.13 видно, що ентропія діагональних ліній зростає під час кризових явищ, що вказує на зростання впливу детермінованих процесів із різним ступенем передбачуваності.

3.2.2.9 Ентропія вертикальних ліній

Ми можемо визначити **Шеннонівську ентропію для розподілу вертикальних структур** (vertical lines entropy) рекурентної діаграми. Імовірність $p(v)$ того, що вертикальна лінія має довжину v, можна оцінити за частотним розподілом $P(v)$ із $p(v) = P(v)/\sum_{v=v_{\min}}^{N} P(v)$. Ентропія Шеннона цієї ймовірності визначається як

$$VLEn = -\sum_{v=v_{min}}^{N} p(v)\ln p(v).$$

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 очікуємо високе значення цього типу рекурентної ентропії. Це означатиме, що ламінарність процесу характеризуються різноманітними періодами довгостроковості пам'яті системи

```
plot_recurrence_measure(measure=ENT_VERT, label='VLEn')
```

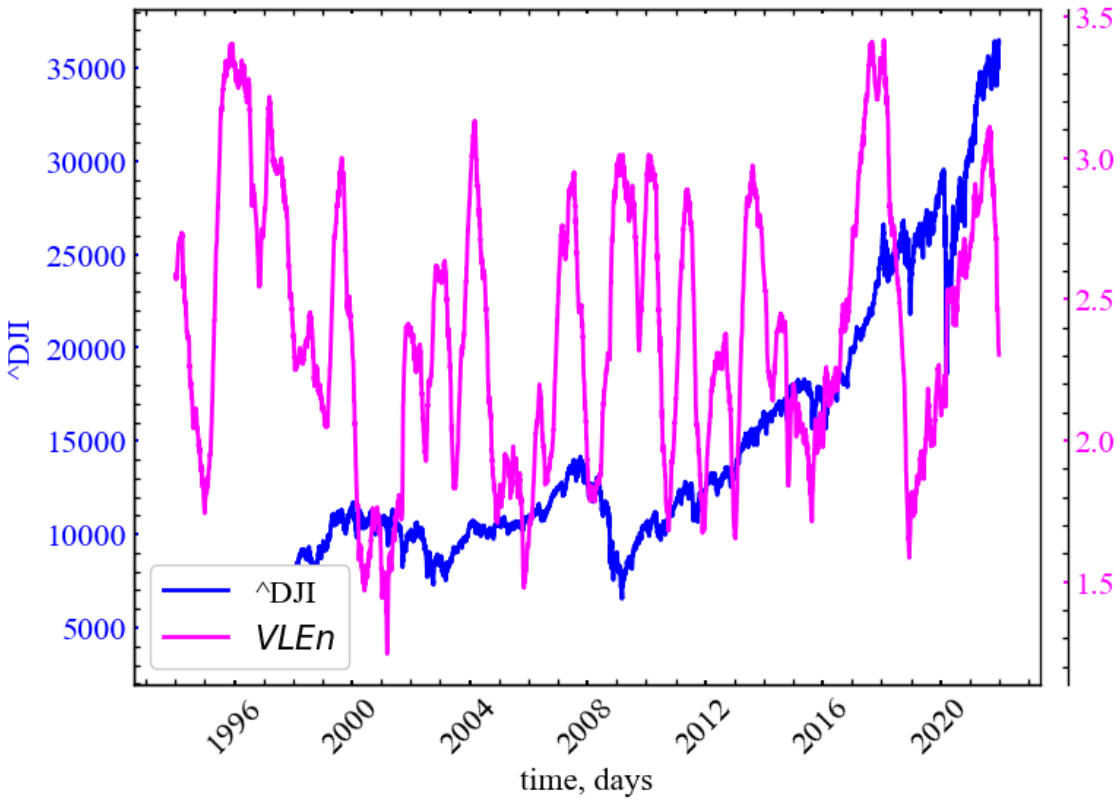


Рис. 3.14: Динаміка індексу Доу-Джонса та ентропії вертикальних ліній

На Рис. 3.14 видно, що ентропія вертикальних ліній починає зростати під час крахових явищ, що вказує на зростання ступеня ламінарності, тобто зростання рівномірності розподілу вертикальних ліній різних довжин.

3.2.2.10 Дивергенція

Показник $L_{\max}$ може надати нам інформацію про максимальний ступінь передбачуваності досліджуваного періоду. Зворотнє значення **максимальної**

довжини діагональних ліній $L_{\max}$ або **дивергенція (розбіжність, divergence)** може вказати нам на швидкість та тривалість розбіжності досліджуваних траєкторій. Даний показник можна визначити як

$$DIV = 1/L_{max}.$$

Дана міра схожа на старший показник Ляпунова [3]. 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).

> 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 одна до одної

```
plot_recurrence_measure(measure=DIV, label='DIV')
```

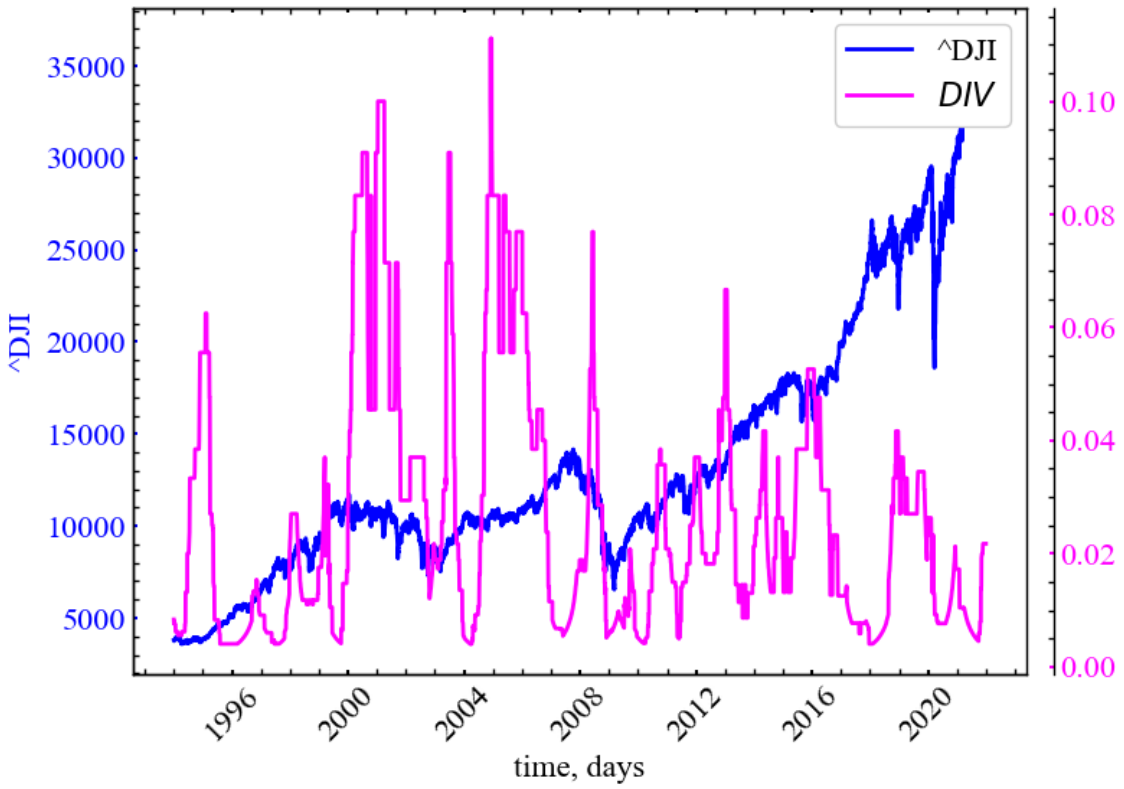


Рис. 3.15: Динаміка індексу Доу-Джонса та дивергенції

Рис. 3.15 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.

3.2.2.11 Дивергенція вертикальних ліній

Зворотнє значення **максимальної довжини вертикальних ліній** V_{max} або **розбіжність вертикальних ліній** (vertical line divergence) можна визначити як

$$VDIV = 1/V_{max}.$$

> 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. Чим вище значення $VDIV$, тим швидше система виходить із ламінарного стану і навпаки

```
plot_recurrence_measure(measure=VERT_DIV, label='VDIV')
```

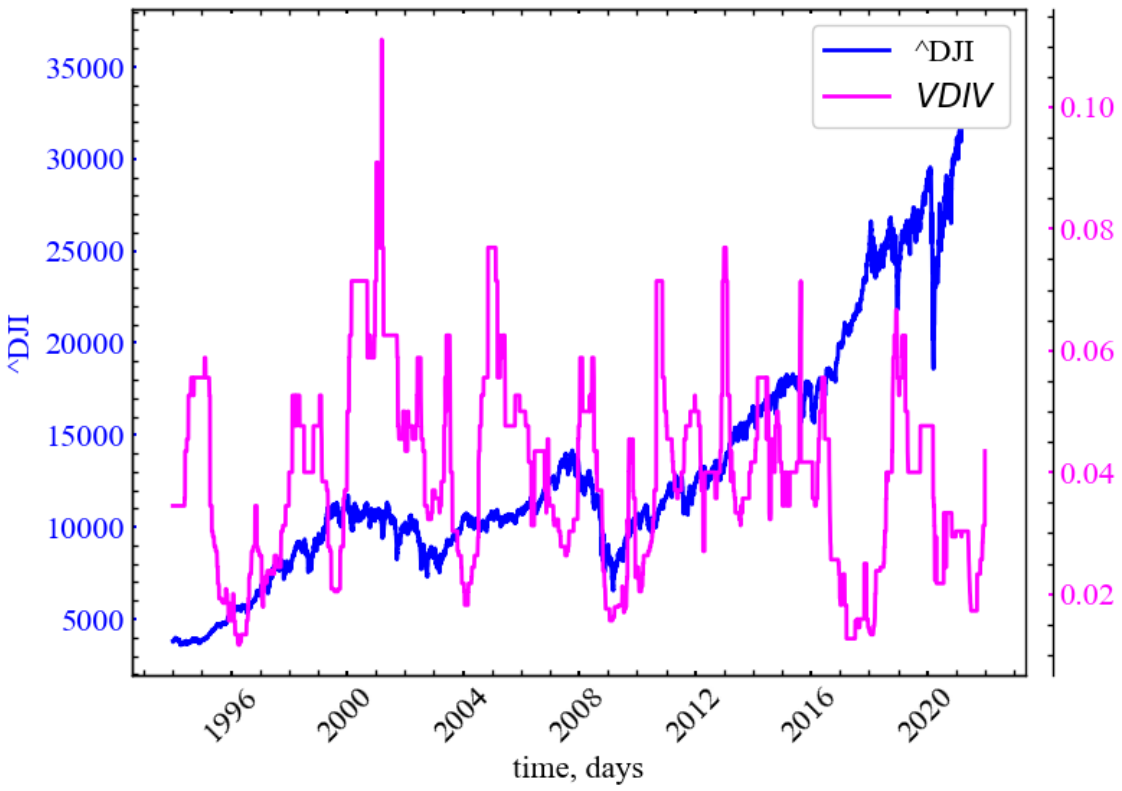


Рис. 3.16: Динаміка індексу Доу-Джонса та дивергенції вертикальних ліній

На даному рисунку (Рис. 3.16) видно, що періоди криз характеризуються спадом вертикальної дивергенції, тобто зростанням кількості вертикальних структур, що характеризують ще більший ступінь ламінарності станів.

3.2.2.12 Дивергенція білих вертикальних ліній

Зворотнє значення **максимальної довжини білих вертикальних ліній** (WVL_{max}) можна охарактеризувати як **дивергенцію білих вертикальних ліній** (white vertical lines divergence). Її можна визначити наступним чином:

$$WVDIV = 1/WVL_{max}.$$

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')
```

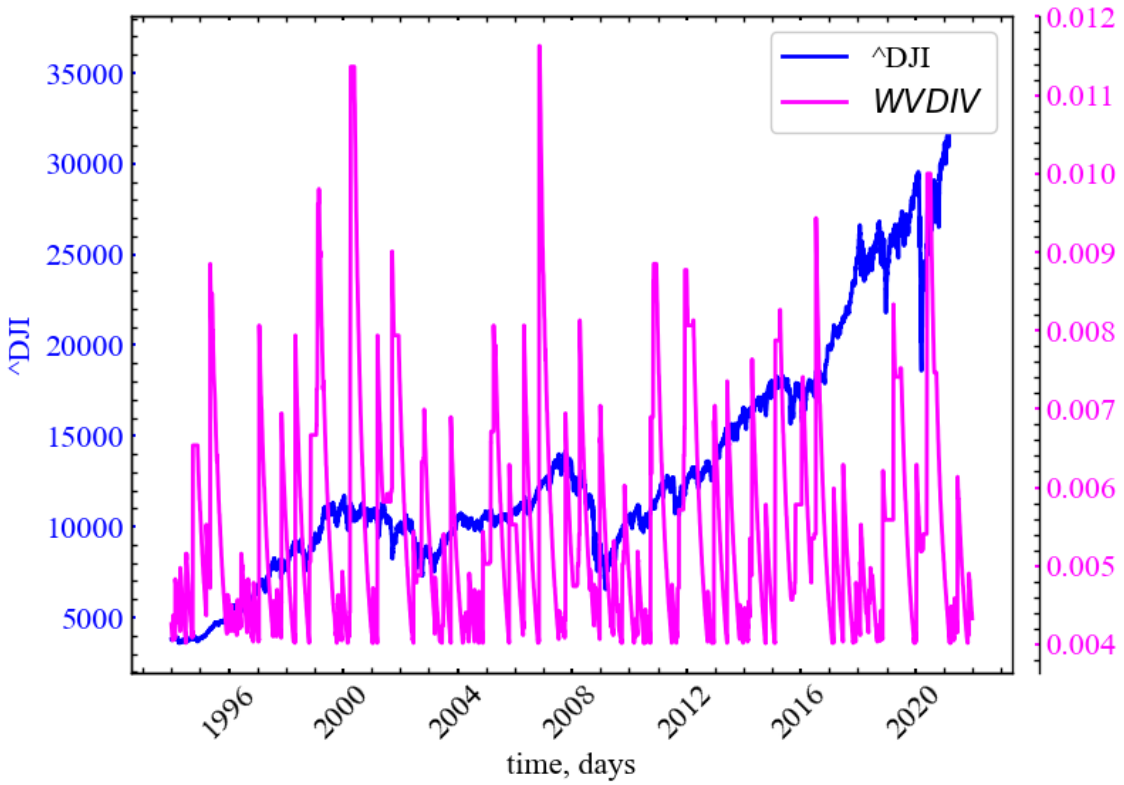


Рис. 3.17: Динаміка індексу Доу-Джонса та дивергенції білих вертикальних ліній

На Рис. 3.17 видно, що дивергенція білих вертикальних ліній представляє доволі зашумлену динаміку, а тому не може бути використана в якості ефективного індикатора кризових явищ.

3.2.2.13 Ентропія білих вертикальних ліній

Імовірність $p(\omega)$ того, що біла вертикальна лінія має довжину ω, можна оцінити за частотним розподілом $P(\omega)$ із $p(\omega) = P(\omega) / \sum_{\omega=\omega_{\min}}^{N} P(\omega)$. **Ентропія Шеннона ймовірності появи білих вертикальних ліній** (white vertical lines entropy) визначається як

$$WVertEn = -\sum_{\omega=\omega_{min}}^{N} p(\omega) \ln p(\omega),$$

де ω_{min} — мінімальна довжина білої вертикальної лінії.

```
plot_recurrence_measure(measure=ENT_WHITE_VERT, label='WVertEn')
```

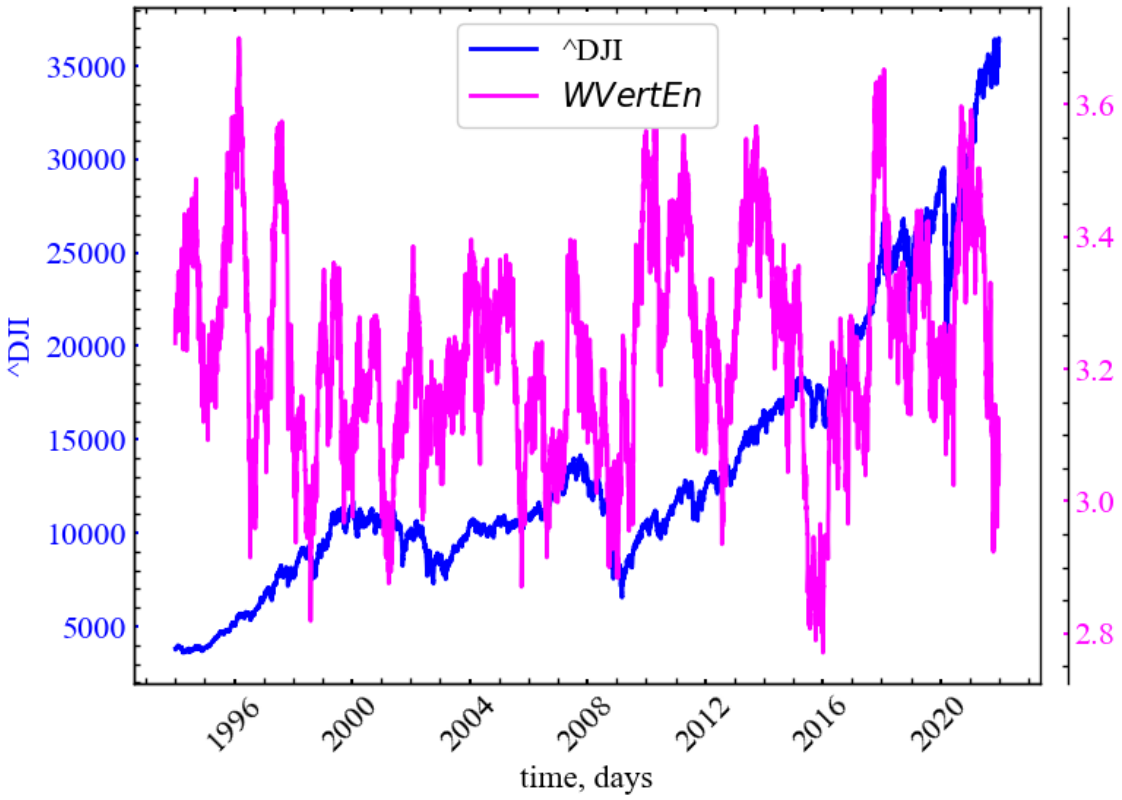


Рис. 3.18: Динаміка індексу Доу-Джонса та ентропії білих вертикальних ліній

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

3.2.2.14 Співвідношення частоти рекурентності до детермінізму (DET/RR)

Співвідношення між DET **і** $RR(RATIO)$ можна використовувати для виявлення прихованих фазових переходів у системи:

$$RATIO_1 = DET/RR = N^2 \cdot \left(\sum_{l=l_{min}}^{N} l \cdot P(l) \right) \Big/ \left(\sum_{l=1}^{N} l \cdot P(l) \right)^2 .$$

```
plot_recurrence_measure(measure=RATIO_DET_REC, label='RATIO_1')
```

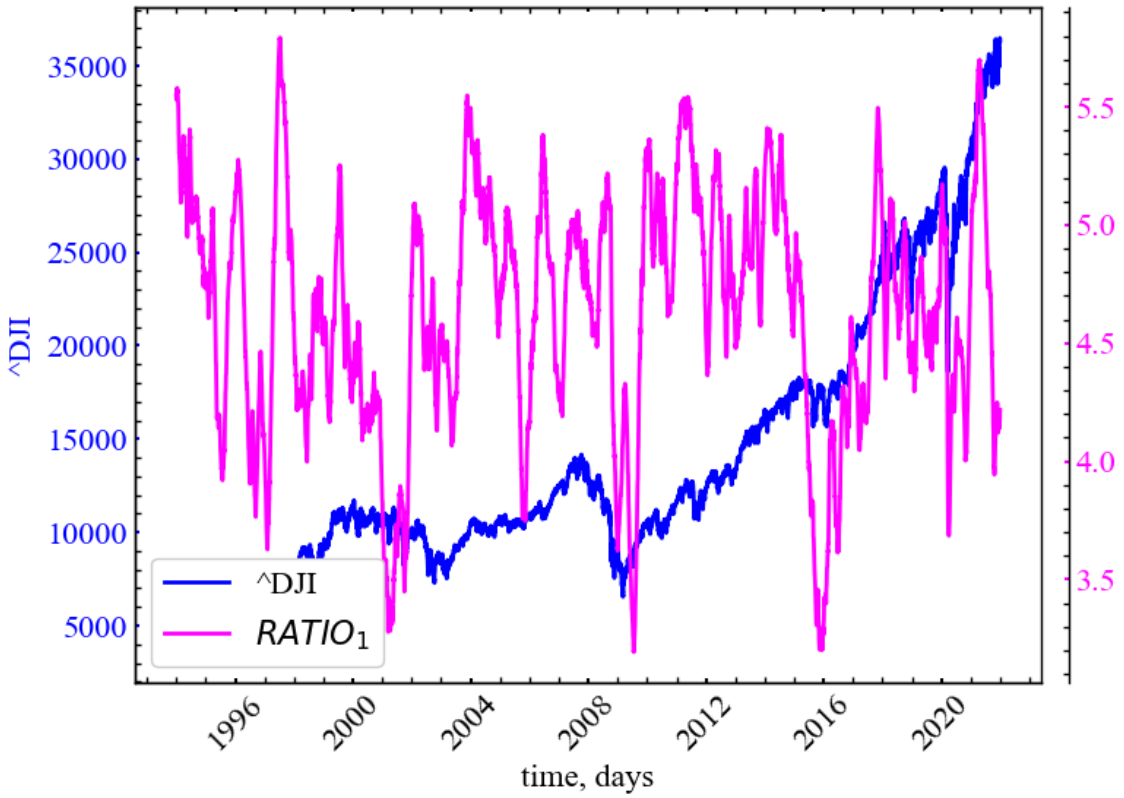


Рис. 3.19: Динаміка індексу Доу-Джонса та співвідношення між мірою передбачуваності та рекурентності

Даний показник спадає під час кризових явищ фондового ринку. Це говорить про те, що має зростати загальна щільність рекурентних точок, як ізольованих, так і всього розподілу вертикальних структур. У кризові періоди RR є вищою за DET.

3.2.2.15 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:

$$RATIO_2 = LAM/DET.$$

```
plot_recurrence_measure(measure=RATIO_LAM_DET, label='RATIO_2')
```

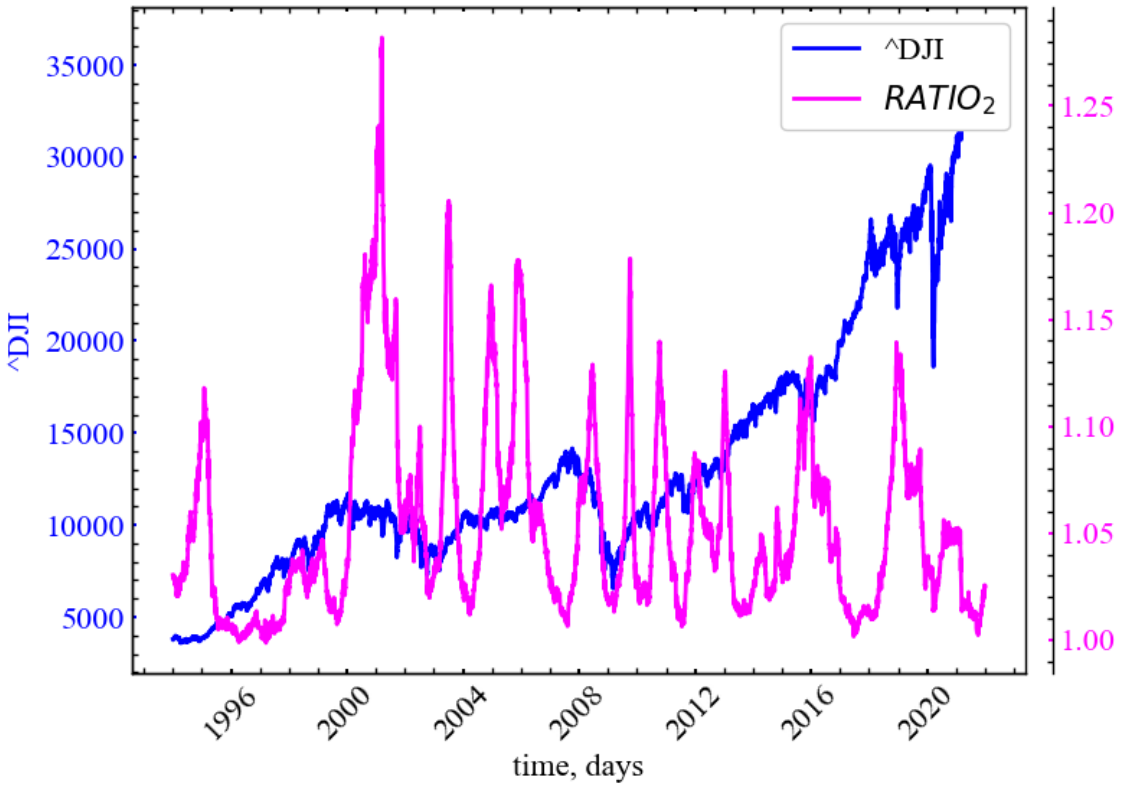


Рис. 3.20: Динаміка індексу Доу-Джонса та співвідношення між мірою ламінарності та детермінізмом

Якщо виходити з динаміки показника $RATIO_2$, можна сказати, що загальний ступінь детермінізму починає переважати над ламінарністю під час кризових явищ.

3.3 Висновок

У даній лабораторній роботі було представлено кількісні рекурентні показники для дослідження еволюції системи. Дані міри було застосовано до часового ряду, що представляє ціни закриття фондового індексу Доу Джонса. Було продемонстровано, що кількісні показники здатні виявляти переходи між хаотичними та періодичними станами (і навпаки), дозволяють ідентифікувати ламінарні стани (хаос-хаос переходи), стани детермінованості й час до настання стану передбачуваності. За результатами представлених показників ми можемо сказати, що досліджувані крахові та передкрахові події характеризуються зростанням рекурентності, і подібного роду поведінка може бути використана в якості передвісника можливих кризових явищ.

3.4 Завдання для самостійної роботи

1. Виберіть за рекомендацією викладача свій варіант часового ряду
2. Проведіть дослідження динаміки кількісних мір рекурентності згідно інструкції
3. Зробити висновки

4. Лабораторна робота № 4

Тема. Інформаційні методи оцінки складності

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

4.1 Теоретичні відомості

4.1.1 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 [29].

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 [30].

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) D. Для даного слова x розглянемо всі його описи, тобто всі слова y, для яких $D(y)$ визначене й рівне x. Довжину найкоротшого з них $l(y)$ і називають колмогорівською складністю слова x при даному способі опису D:

$$KS_D(x) = \min\{l(y) \mid D(y) = x\},$$

де $l(y)$ позначає довжину слова y. Індекс D підкреслює, що визначення залежить від вибору способу D.

Можна показати, що існують оптимальні способи опису. Спосіб опису тим кращий, чим він коротший. Тому природно дати таке визначення: спосіб D_1 не гірше за спосіб D_2, якщо $KS_{D_1}(x) \leq KS_{D_2}(x) + c$ при деякому c і при всіх x.

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. Також можна розглядати алгоритмічну складність як мінімальний час (або інші обчислювальні ресурси), необхідний для виконання цієї задачі на комп'ютері. А

ще ми можемо говорити про комунікаційну складність завдань, в яких задіяно більше одного процесора: це кількість бітів, які потрібно передати при розв'язанні цього завдання [31,32]. На жаль, це визначення чисто умоглядне. Надійного способу однозначно визначити цю програму не існує. Але є алгоритми, які фактично якраз і намагаються обчислити колмогорівську складність тексту [33] і ентропію [34].

4.1.2 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 (Lempel and Ziv, LZ) [35]. **Складність Лемпеля-Зіва** (Lempel-Ziv complexity, LZC) є класичною мірою, яка для ергодичних джерел пов'язує поняття складності (у розумінні Колмогорова-Чайтіна) та швидкості ентропії [36,37]. Для ергодичного динамічного процесу кількість нової інформації, отриманої за одиницю часу (швидкість ентропії), може бути оцінена шляхом вимірювання здатності цього джерела генерувати нові патерни. Завдяки простоті методу LZC, швидкість ентропії може бути оцінена з однієї дискретної послідовності вимірювань з низькими обчислювальними витратами [38]. 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).
2. Копіювання "готового" фрагмента з передісторії (тобто з уже синтезованої частини тексту).

Нехай Σ — скінчений алфавіт, S — текст (послідовність символів), складений з елементів Σ; $S[i]$ — i-й символ тексту; $S[i:j]$ — фрагмент тексту з i-го по j-й символ включно $(i < j)$; $N = |S|$ — довжина тексту S. Тоді схему синтезу послідовності можна представити у вигляді конкатенації

$$H(S) = S[1:i_1]S[i_1+1:i_2]\dots S[i_{k-1}+1:i_k]\dots S[i_{m-1}+1:N], \qquad (4.1)$$

де $S[i_{k-1}+1:i_k]$ — фрагмент S, породжуваний на k-му кроці, а $m = m_H(S)$ — число кроків процесу. З усіляких схем породження S обирається мінімальна за числом кроків. Таким чином, складність послідовності S за LZ

$$c_{LZ}(S) = \min_H\{m_H(S)\}.$$

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

$$l_{j(k)} = i_k - i_{k-1} - 1 = \max_{j \le i_{k-1}} \{l_j : S[i_{k-1}+1 : i_{k-1}+l_j]\} = S[j : j+l_j-1], \tag{4.2}$$

а сам k-й компонент складнісного розкладання (4.1) можна записати у вигляді

$$S[i_{k-1}+1 : i_k] = \begin{cases} S[j(k) : j(k)+l_{j(k)}-1], & \text{якщо } j(k) \neq 0, \\ S[i_{k-1}+1], & \text{якщо } j(k) = 0. \end{cases} \tag{4.3}$$

Випадок $j(k) = 0$ відповідає ситуації, коли в позиції $i_{k-1}+1$ стоїть символ, який раніше не зустрічався. При цьому ми застосовуємо операцію генерації символу.

Будемо знаходити складність за LZ для часового ряду, який являє собою, наприклад, щоденні значення фінансового індексу. Для дослідження динаміки LZ та порівняння з іншими складними системами 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, що 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 [39–41]:

$$ret = \begin{cases} 0, & ret_t < \langle ret \rangle, \\ 1, & ret_t > \langle ret \rangle. \end{cases} \tag{4.4}$$

Також можна визначити так звану **пермутаційну складність Лемпеля-Зіва** (permutation Lempel-Ziv complexity, PLZC) [42,43]. У даному випадку би будемо опиратись на процедуру реконструкції фазового простору, що згадувалась у лабораторних 2 і 3. Згідно пермутаційній процедурі ми будемо брати фрагмент ряду довжини m, що слугує розмірністю реконструйованого атрактора, та замінювати кожне значення ряду його порядковим індексом. На Рис. 4.1 представлено часовий ряд та його можливі порядкові шаблони:

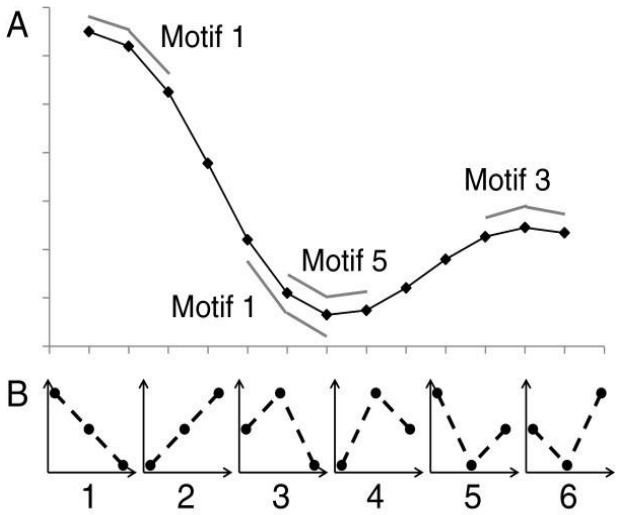


Рис. 4.1: Фрагмент часового ряду (a) та 6 можливих порядкових шаблонів, що можуть зустрічатись у цьому сигналі (b) [44]

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 n алгоритмічна складність обчислюється за виразом $LZC_r = n/\log(n)$. Тоді відносна алгоритмічна складність знаходиться як відношення отриманої складності до складності випадкової послідовності: $LZC = LZC/LZC_r$.

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 мономасштабні розрахунки алгоритмічної складності можуть бути неприйнятними і призводити до помилкових висновків.

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

4.1.3 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 усередненні послідовних відліків ряду в межах вікон, що не перетинаються, а розмір яких τ — збільшується при переході від масштабу до масштабу. Кожен елемент “гранульованого” часового ряду y_j^τ знаходиться у відповідності до виразу [45]:

$$y_j^\tau = \frac{1}{\tau} \sum_{i=(j-1)\tau+1}^{j\tau} x_i\,,\ 1 \le j \le N/\tau,$$

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

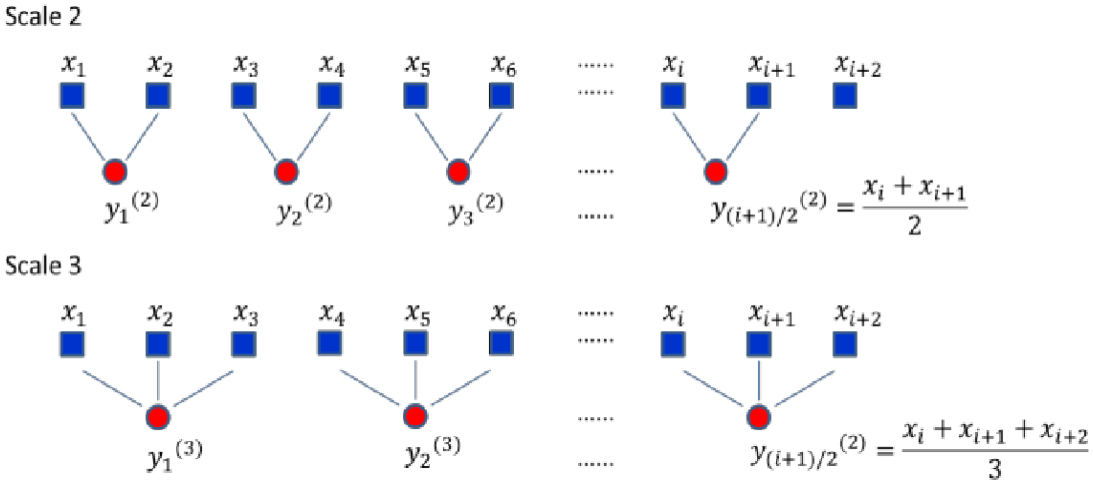


Рис. 4.2: Схематична ілюстрація процесу грубого дроблення (“грануляції”) початкового часового ряду для масштабів 2 і 3

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;

- **delay** (*int*) — часова затримка, τ. Використовується лише тоді, коли `permutation=True`;
- **dimension** (*int*) — розмірність вкладень, d_E. 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 застосування 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, містить додаткову інформацію про параметри, що 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;
- **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 d_E;
- **tolerance** (*float*) — поріг пропускання ε;
- **method** (*str*) — яку версію мультимасштабного показника обчислювати. Переважна кількість показників за цим методом відповідають 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 щодо використаних для обчислення мультимасштабного показника параметрів. Значення показника, що відповідають кожному фактору `"Scale"`, зберігаються під ключем `"Value"`.

4.1.4 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 (Shannon entropy) — це статистичний квантифікатор, який широко використовується для характеристики складних процесів. Поняття ентропії було використано Шенноном в теорії інформації для передачі даних [34].

Ентропія — це міра невизначеності та випадковості системі. Якщо припустити, що всі наявні дані належать до одного класу, то неважко передбачити клас нових даних. Невизначеність, що виникає, коли подія E відбувається з ймовірністю p, можна позначити як $S(p)$. Якщо ймовірність появи класу дорівнює 1, тоді ентропія мінімальна, $S(1) = 0$. Відповідно до концепції Шеннона, якщо у нас наявні ймовірності реалізації певної події $p_1, p_2, p_3, \ldots, p_n$, на виході отримується кількість інформації, що необхідна для опису цієї події. Тоді, Шеннонівська ентропія може бути визначена як

$$S = -\sum_{i=1}^{n} p_i \ln p_i.$$

Синтаксис методу для розрахунку Шеннонівської ентропії має вигляд:

```
entropy_shannon(signal=None, base=2, symbolize=None, show=False, freq=None, **kwargs)
```

Параметри:

- **signal** (*Union[list, np.array, pd.Series]*) — сигнал;

- **base** (*float*) — основа логарифму (за замовчуванням дорівнює 2). `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 (вважається, що вхідні дані вже є дискретними);
- **show** (*bool*) — якщо значення `True`, виводить часовий ряд, де кожне значення розфарбоване у відповідності до класу до якого воно належить;
- **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 обчислення Шеннонівської ентропії.

4.1.5 Інформація Фішера

Інформацію Фішера (Fisher information, FI) було введено Р. А. Фішером у 1922 році як міру "внутрішньої точності" в теорії статистичних оцінок [46]. Вона є центральною для багатьох статистичних застосувань, 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 системи "про себе". Він базується на розкладанні за сингулярними значеннями реконструйованого фазового простору. Величина FI зазвичай антикорельована з іншими показниками складності (чим більше інформації система приховує про себе, тим більш передбачуваною і, відповідно, менш складною вона є).

FI можна визначити, використовуючи метод `fisher_information()` бібліотеки `neurokit2`. Її синтаксис виглядає наступним чином:

`fisher_information(signal, delay=1, dimension=2)` з визначеними вже раніше параметрами

Повертає:

- **fi** (*float*) — обчислена міра FI;
- **info** (*dict*) — словник, що містить додаткову інформацію про параметри обчислення FI.

4.1.6 Складність Хьорта (Hjorth's complexity) та його параметри

Параметри Хьорта — це показники статистичних властивостей, які спочатку були введені Хьортом [47] для опису загальних характеристик сигналів електроенцифалограми. Параметрами є активність, рухливість і складність:

1. Параметр **активності** ($Activity$) — це просто дисперсія сигналу, яка відповідає середній потужності сигналу (якщо його середнє значення дорівнює 0):

$$Activity = \sigma_{signal}^2.$$

2. Параметр **рухливості** ($Mobility$) являє собою середню частоту або частку середньоквадратичного відхилення спектра потужності. Він визначається як квадратний корінь з дисперсії першої похідної сигналу, поділений на дисперсію сигналу:

$$Mobility = \frac{\sigma_{dd}/\sigma_d}{Complexity}.$$

3. Параметр **складності** ($Complexity$) дає оцінку смуги пропускання сигналу, що вказує на схожість форми сигналу з чистою синусоїдою (для якої значення сходиться до 1). Іншими словами, це характеристика "надмірної деталізації" по відношенню до "найм'якшої" можливої форми кривої. Параметр "Складність" визначається як відношення рухливості першої похідної сигналу до рухливості самого сигналу:

$$Complexity = \sigma_d/\sigma_{signal},$$

де d та dd представляють перші та другі похідні сигналу, відповідно.

Бібліотека `neurokit2` представляє метод для отримання відповідних показників. Її синтаксис:

```
complexity_hjorth(signal)
```

Повертає:

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

4.1.7 Час декореляції

Час декореляції (decorrelation time, DT) визначається як час (у відліках) першого перетину нуля функції автокореляції. Коротший DT відповідає менш корельованому сигналу. Наприклад, зменшення DT в сигналах електроенцифалограми спостерігається перед нападами, що пов'язано зі зменшенням потужності низьких частот [48].

Бібліотека `neurokit2` представляє функціонал для визначення DT, а саме метод `complexity_decorrelation()`. Її синтаксис:

```
complexity_decorrelation(signal)
```

Повертає:

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

4.1.8 Відносна грубість (нерівність, шорсткість)

Відносна шорсткість (relative roughness, RLR) — це відношення локальної дисперсії (автоковаріації з лагом 1) до глобальної дисперсії (автоковаріації з лагом 0), яке можна використовувати для класифікації різних "шумів". Цей показник також можна використовувати як індекс застосовності фрактального аналізу (показники фрактальності будуть описані в наступних роботах) [49].

Синтаксис даного методу в бібліотеці `neurokit2` виглядає наступним чином:

```
complexity_relativeroughness(signal, **kwargs)
```

Повертає:

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

4.1.9 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. Інші статистичні методи вимірюють нелінійні зв'язки, такі як, наприклад, **взаємна інформація** (mutual information, MI) [50].

MI між двома випадковими величинами вимірює нелінійний зв'язок між ними. Крім того, вона показує, **яку кількість інформації можна отримати з випадкової величини**, спостерігаючи за іншою випадковою величиною.

Вона тісно пов'язана з поняттям ентропії. Тобто, зменшення невизначеності випадкової величини корелює з отриманням інформації з іншої випадкової величини. І високе значення взаємної інформації вказує на помітне зменшення невизначеності. Якщо взаємна інформація дорівнює нулю, дві випадкові величини є незалежними.

MI можна розрахувати наступним чином:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} (x,y) \cdot \log[p(x,y)/p(x) \cdot p(y)],$$

де $p(x)$ та $p(y)$ ймовірності спостереження окремо x або y, а $p(x,y)$ ймовірність спостереження одночасно x та y.

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). Взаємна інформація дорівнює нулю, коли дві випадкові величини є строго незалежними.

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

Синтаксис потрібної нам процедури:

```
mutual_information(x, y, method='varoquaux', bins='default', **kwargs)
```

Параметри:

- **x** і **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 описаної в [51];
- **kwargs** — додаткові ключові аргументи для обраного методу.

Повертає:

- *float* — розрахована взаємна інформація.

Існують різноманітні підходи до розрахунку взаємної інформації:

- **nolitsa**: класична взаємна інформація;

- **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 MI є максимальним при певній комбінації кількості бінів.

Розглянемо ефективність використання зазначених показників у якості індикаторів або індикаторів-передвісників крахових подій.

4.2 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

size = 16
params = {
'figure.figsize': (8, 6),           # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                  # розмір фонтів рисунку
'lines.linewidth': 2,               # товщина ліній
'axes.titlesize': 'small',          # розмір титулки над рисунком
'axes.labelsize': size,             # розмір підписів по осям
'legend.fontsize': size,            # розмір легенди
'xtick.labelsize': size,            # розмір розмітки по осі Ox
'ytick.labelsize': size,            # розмір розмітки по осі Ox
"font.family": "Serif",             # сімейство стилів підписів
"font.serif": ["Times New Roman"],  # стиль підпису
'savefig.dpi': 300,                 # якість збережених зображень
'axes.grid': False                  # побудова сітки на самому рисунку
}

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 та криптовалютним ринками. Очевидно, що фондовий індекс S&P 500 має довшу ніж Біткоїн історію. До того ж, криптовалютний ринок працює безперервно на відміну від фондового. 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 Ox
ylabel_1 = symbol_1        # підпис по вісі Oy
```

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 Ox
ylabel_2 = symbol_2        # підпис по вісі Oy
```

> ⚠Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того, з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'           # підпис по вісі Ox
ylabel = symbol                     # підпис по вісі Oy
```

Тепер створимо новий масив даних, що об’єднуватиме в собі значення S&P 500 та BTC за їх спільними датами:

```
# 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
```

Візуалізуємо вилучені дані. 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', rotation=45, **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)
```

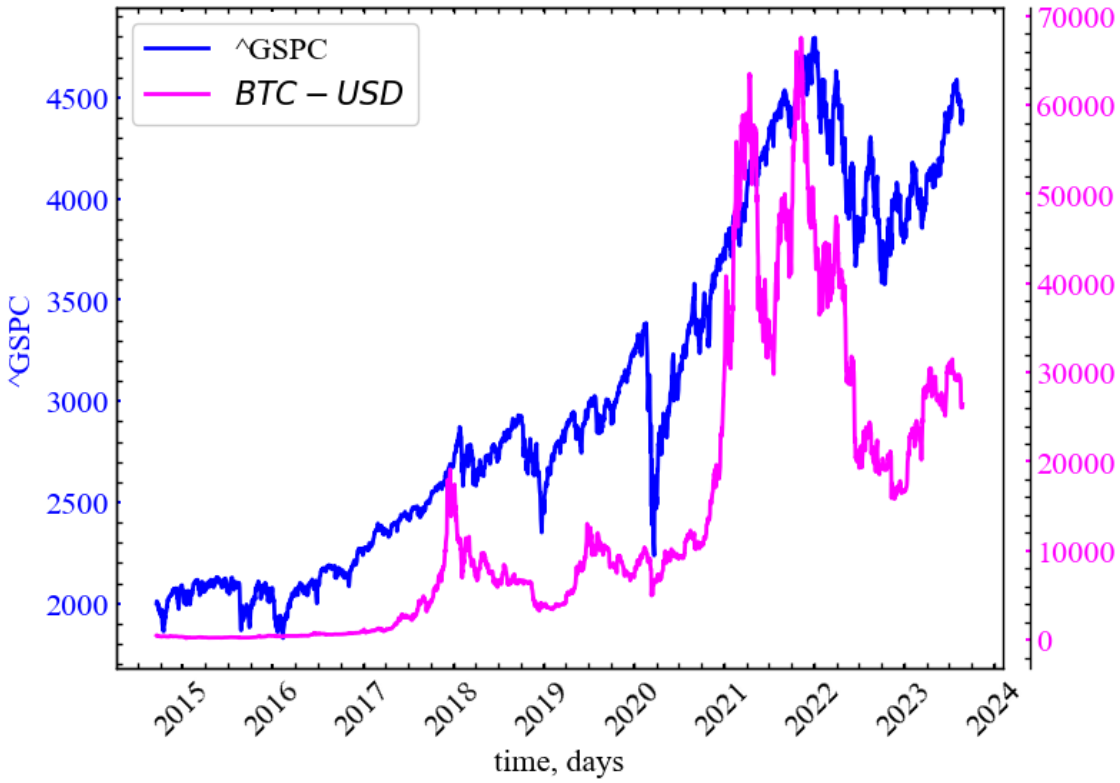


Рис. 4.3: Динаміка індексу S&P 500 та Біткоїна за досліджуваний період

> ⓘ Важливо
>
> Не виконуйте блоки коду, що відповідають секції “Розрахунок взаємної інформації”, якщо ви працюєте з текстовим файлом

4.2.1 Розрахунок взаємної інформації

Розглянемо взаємну інформацію як індикатор нелінійної кореляції між двома фінансовими активами, і спробуємо визначити, чи є між ними “істинний” взаємозв’язок. Виконаємо 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)
```

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(1, 1)
```

```
ax2 = ax.twinx()
ax3 = ax.twinx()
ax2.spines.right.set_position(("axes", 1.03))
ax3.spines.right.set_position(("axes", 1.15))

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', rotation=45, **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();
```

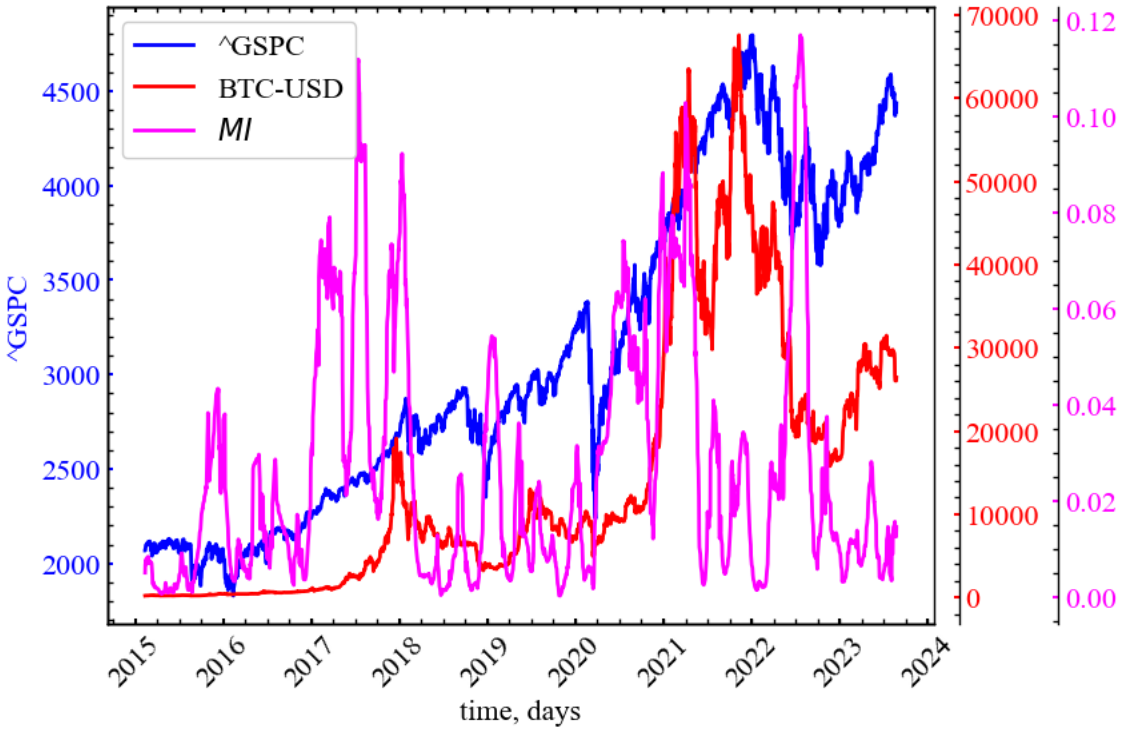


Рис. 4.4: Динаміка індексу S&P 500, Біткоїна та взаємної інформації

Як ми можемо бачити з представленого рисунку, як на фондовому так і криптовалютному ринках дійсно спостерігалися фази зростання взаємної інформації. Найкраще це видно напередодні кризи 2018-го року, під час 2019 року, 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 фінансових ринках.

Як вже зазначалося, окрім обчислення взаємної інформації для двох пар часових сигналів, ми можемо обчислити автовзаємну інформацію.

Для цього 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 inenumerate(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(1, 1)                # 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("Часова затримка")            # Додаємо підпис для вісі Ox
ax.set_ylabel("Авто-взаємна інформація")    # Додаємо підпис для вісі Oy

plt.savefig(f'Automutual information {symbol_1}.jpg')  # Зберігаємо графік
plt.show();# Виводимо графік
```

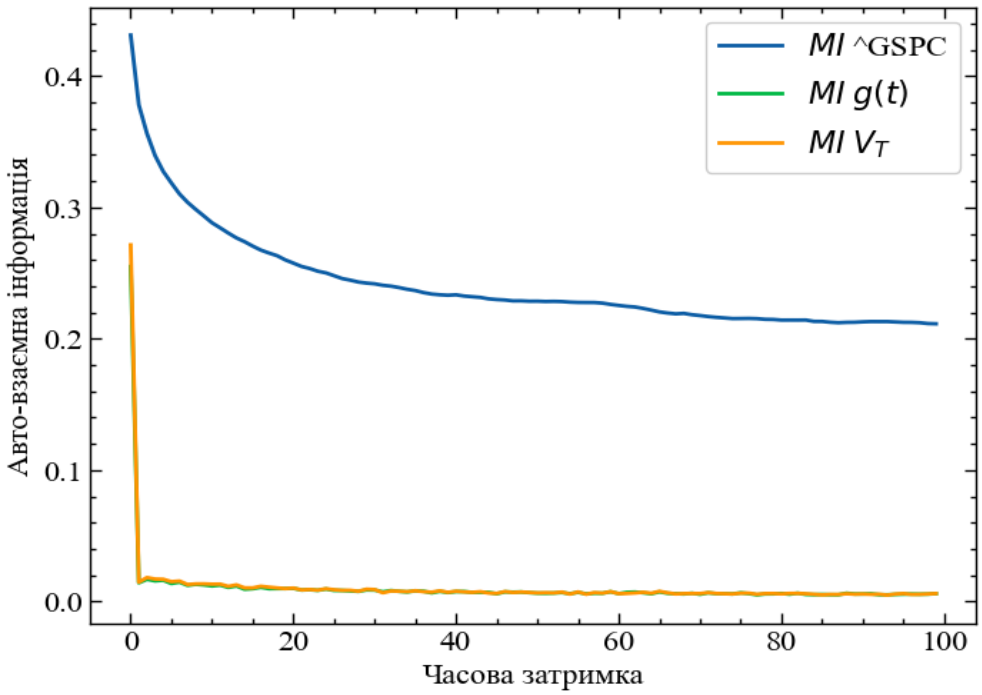


Рис. 4.5: Зміна з часом автовзаємної інформації для вихідного ряду x, нормалізованих прибутковостей g та модулів $|g|$фондового індексу S&P 500

```
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("Часова затримка")                # Додаємо підпис для вісі Ox
ax.set_ylabel("Авто-взаємна інформація")        # Додаємо підпис для вісі Oy

plt.savefig(f'Automutual information {symbol_2}.jpg')  # Зберігаємо графік
plt.show();                                            # Виводимо графік
```

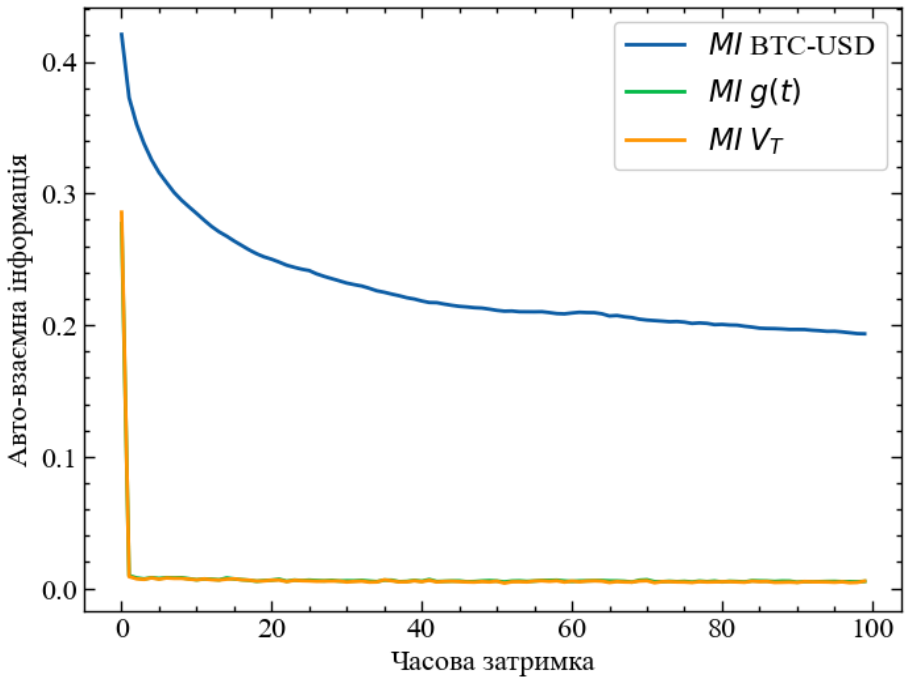


Рис. 4.6: Зміна з часом автовзаємної інформації для вихідного ряду x, нормалізованих прибутковостей g та модулів $|g|$ криптовалютного індексу BTC

Очевидно, 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.

4.2.2 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)
```

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(1, 1)

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', rotation=45, **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();
```

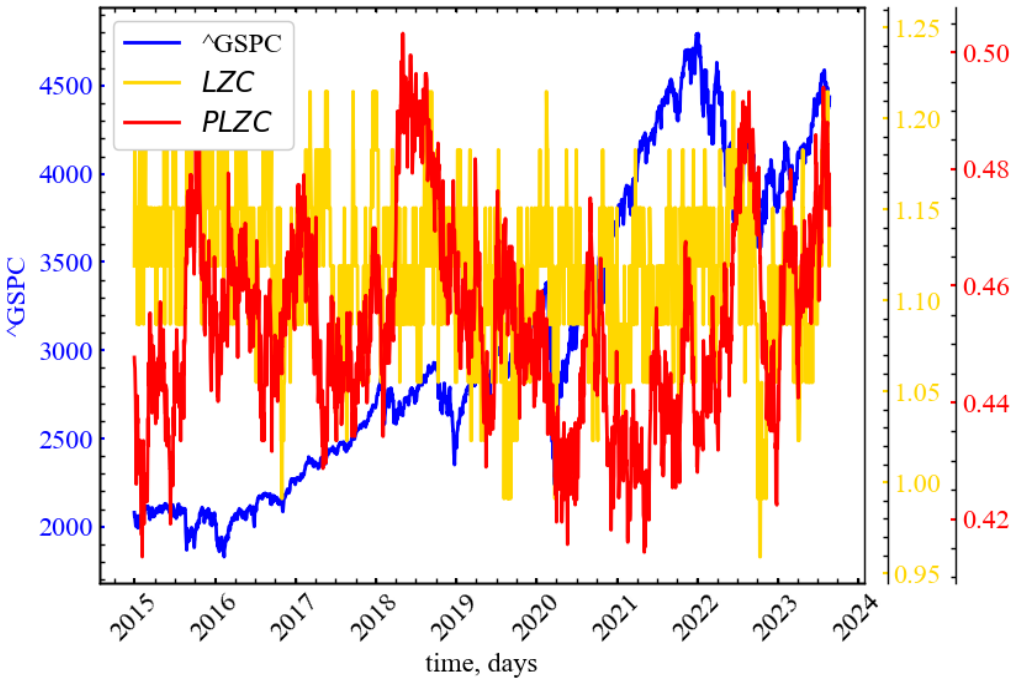


Рис. 4.7: Динаміка індексу S&P 500, класичної мономасштабної складності Лемпеля-Зіва та її пермутаційного аналогу

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

4.2.3 Обчислення мультимасштабної складності Лемпеля-Зіва

Розрахуємо віконну динаміку мультимасштабних показників Лемпеля-Зіва. Ми повертаємо сумарну складність Лемпеля-Зіва за всіма масштабами:

```
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)

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(1, 1)

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', rotation=45, **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();
```

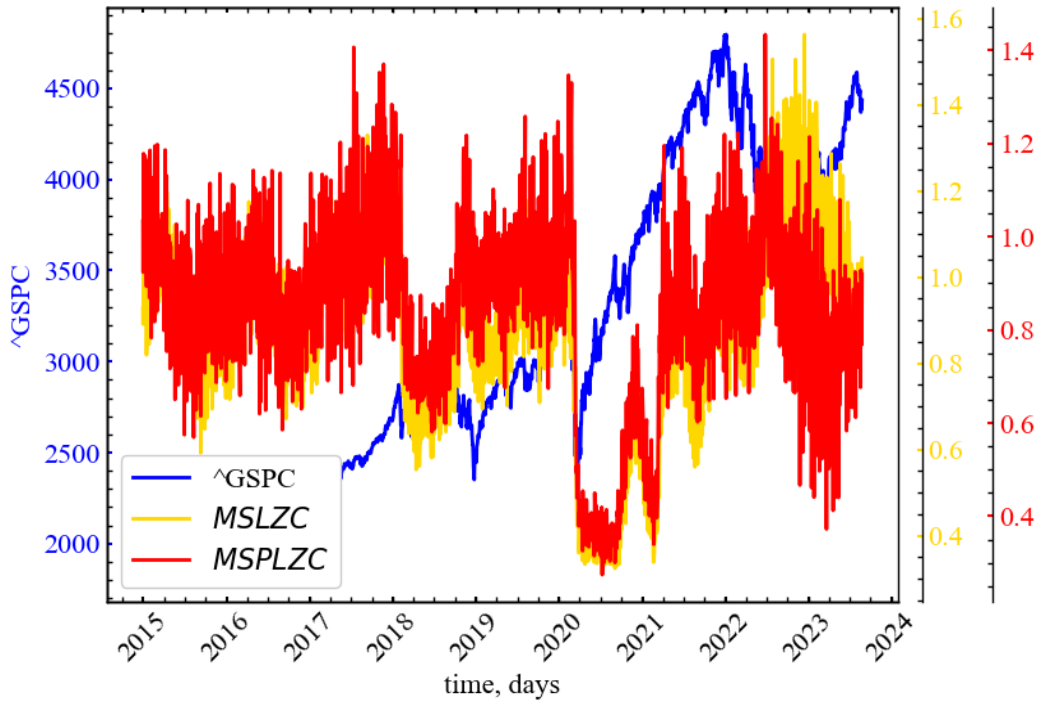


Рис. 4.8: Динаміка індексу S&P 500, класичної мультимасштабної складності Лемпеля-Зіва та її пермутаційного аналогу

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.

4.2.4 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)

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)
```

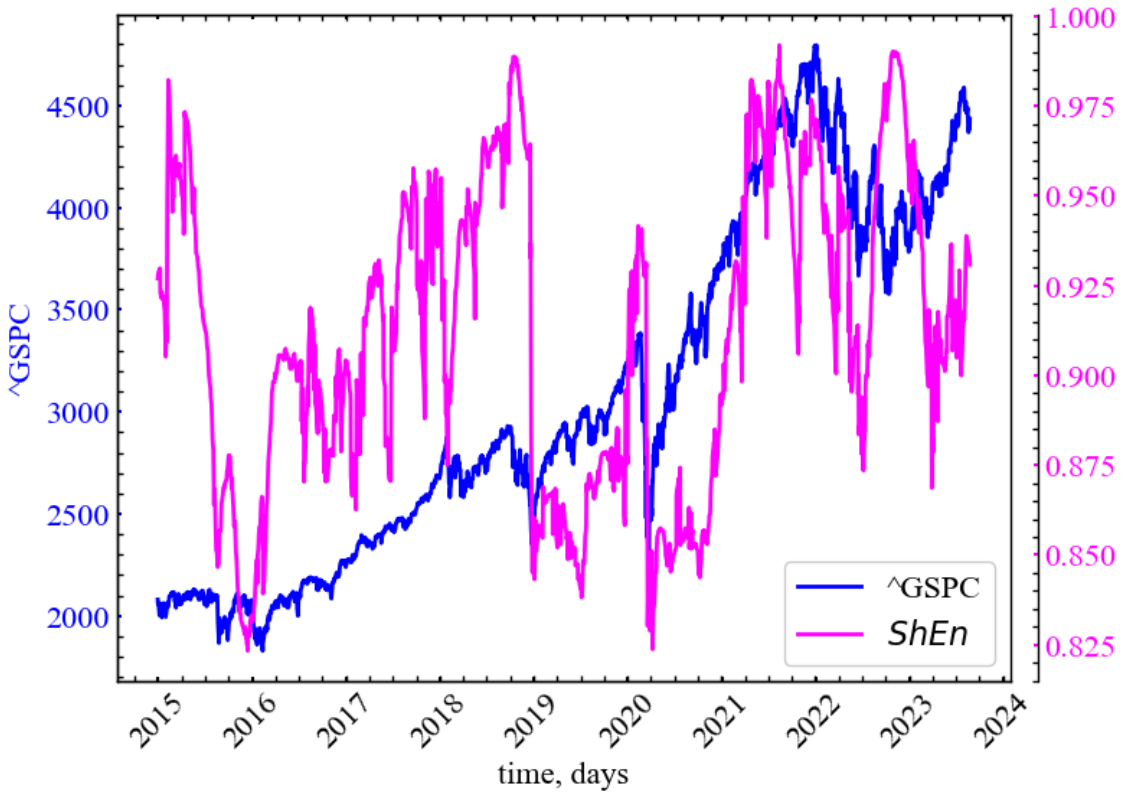


Рис. 4.9: Динаміка індексу S&P 500 та ентропії Шеннона

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

4.2.5 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)

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)
```

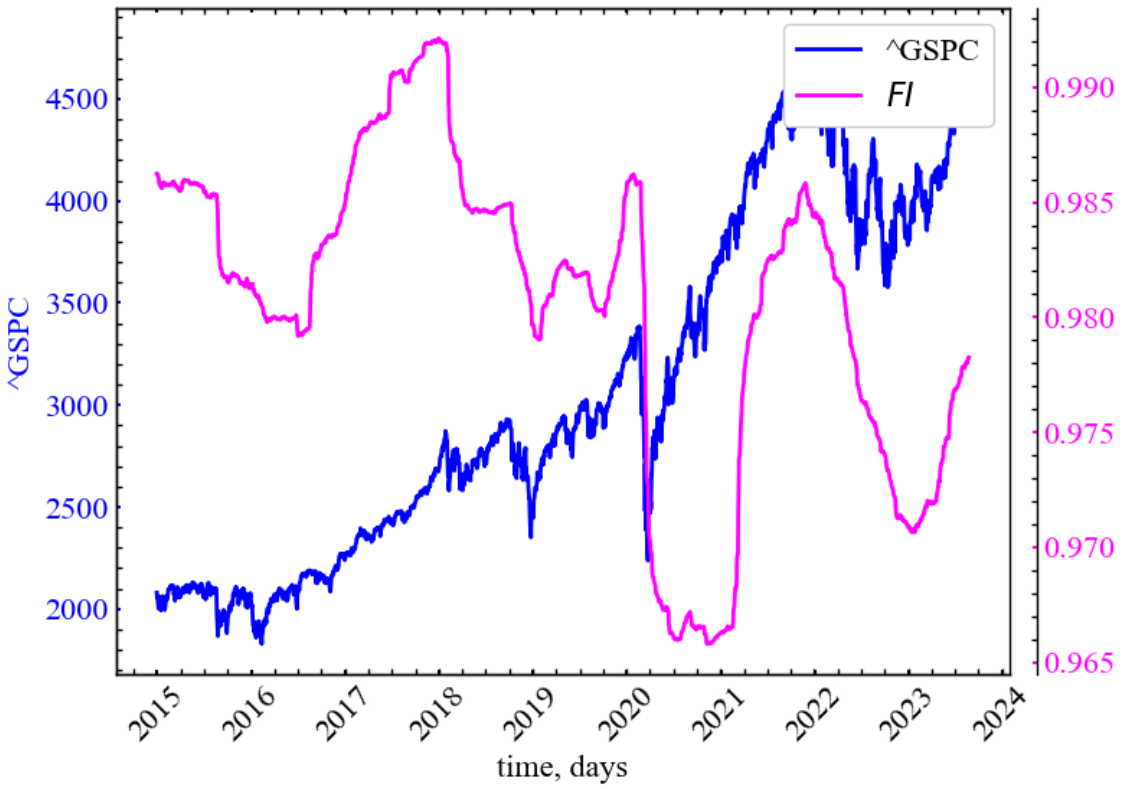


Рис. 4.10: Динаміка індексу S&P 500 та інформаційного показника Фішера

З Рис. 4.10 видно, що показник Фішера спадає у кризові та передкризові періоди, що говорить про спад кількості необхідної для опису самоорганізованої динаміки фінансових криз інформації, зростання корельованості між діями трейдерів на ринку.

4.2.6 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)

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)
```

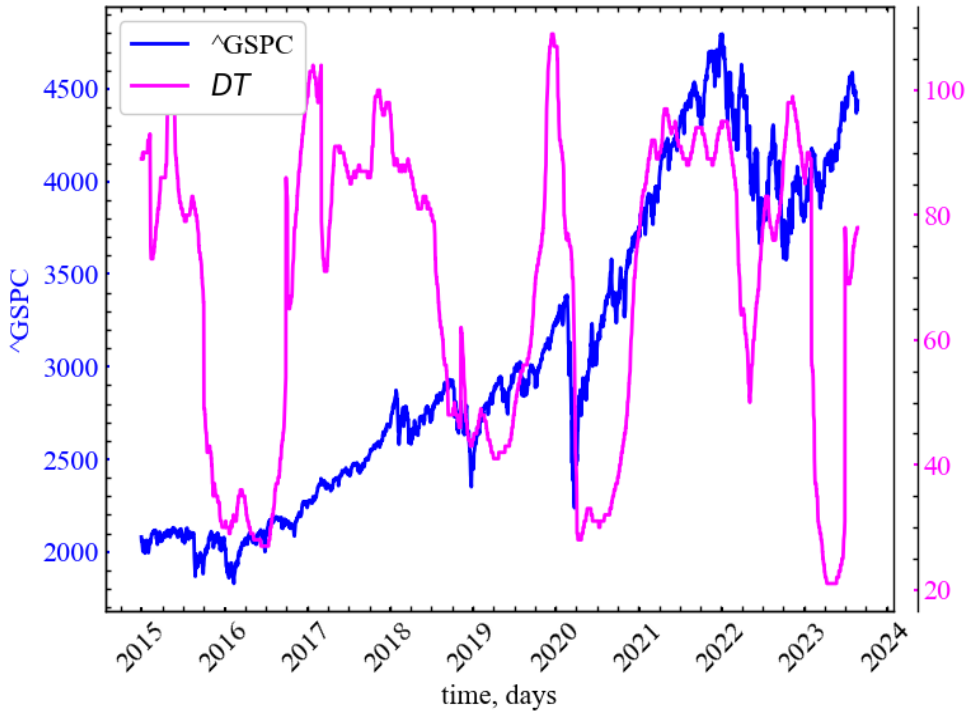


Рис. 4.11: Динаміка індексу S&P 500 та часу декореляції

Час декореляції зростає у передкраховий період, що вказує на посилення кореляції системи в цей період.

4.2.7 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)

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)
```

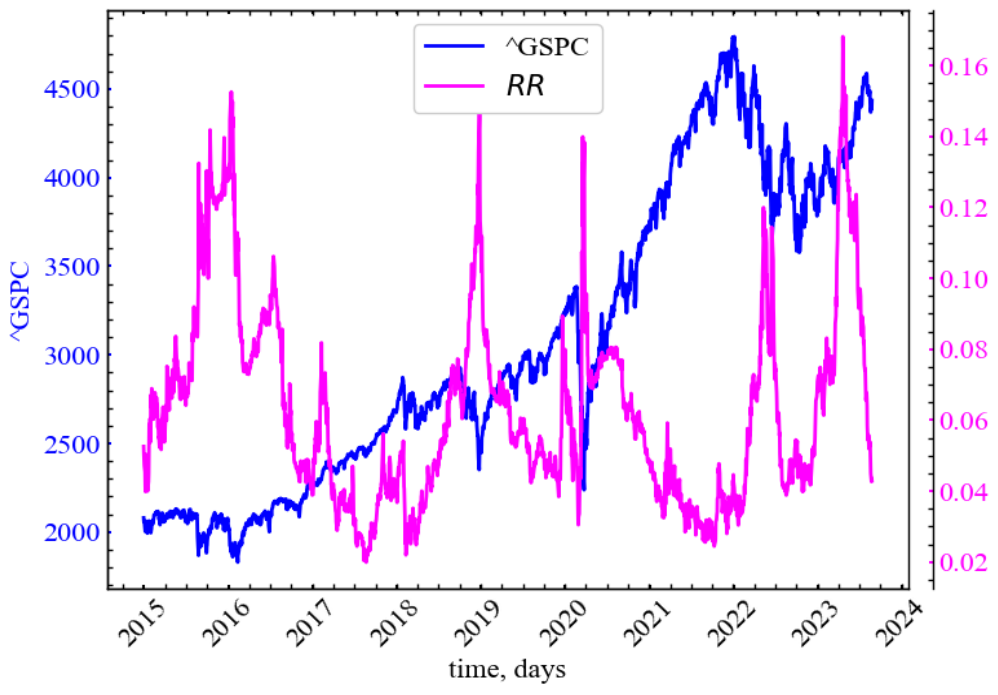


Рис. 4.12: Динаміка індексу S&P 500 та показника відносної шорсткості

Показник відносної шорсткості демонструє, що крахові події як, наприклад, у 2015, 2016, 2019, 2020 та 2023 роках характеризуються зростанням шорсткості. Подібного роду поведінка є індикатором зростання шумової активності ринку: кореляційних характеристик та загальної варіації ринку в цілому.

4.2.8 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)

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(1, 1)

ax2 = ax.twinx()
ax3 = ax.twinx()
ax4 = ax.twinx()

ax2.spines.right.set_position(("axes", 1.03))
ax3.spines.right.set_position(("axes", 1.16))
ax4.spines.right.set_position(("axes", 1.24))
```

```
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)
ax.tick_params(axis='x', rotation=45, **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)
ax4.legend(handles=[p1, p2, p3, p4])

plt.savefig(f"hjorth_name={symbol_1}_ret={ret_type}_wind={window}_step={tstep}.j
pg")
plt.show();
```

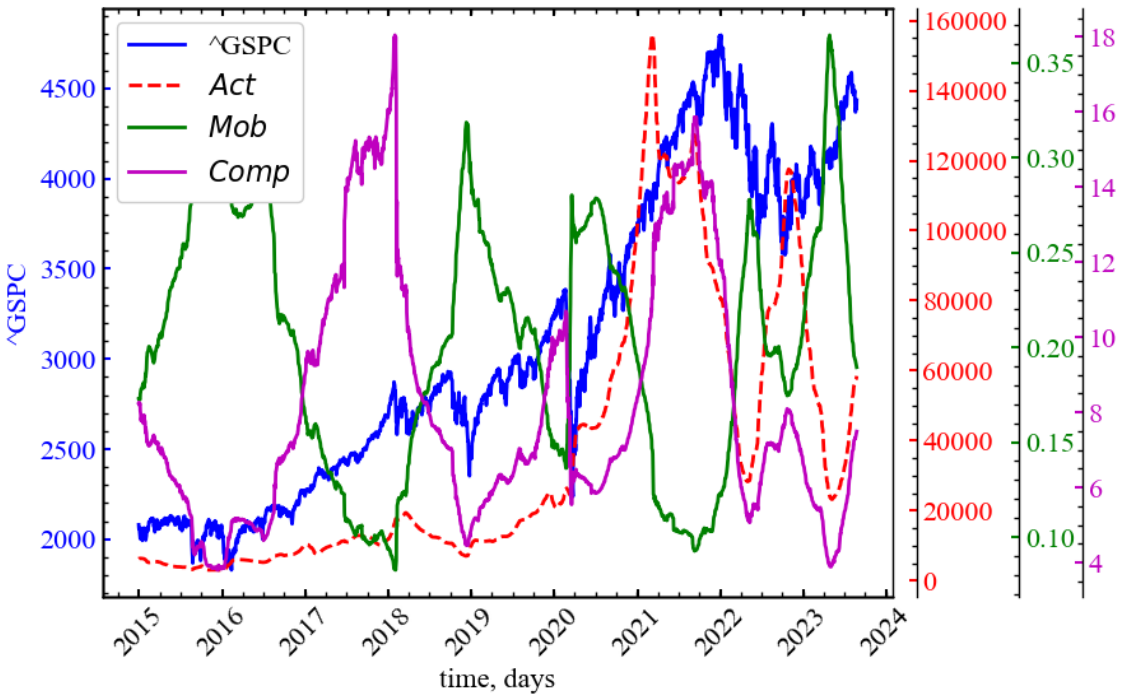


Рис. 4.13: Динаміка індексу S&P500 наряду з показниками активності,

мобільності та складності Хьорта

Очевидно, що параметр активності (Act) представляється найменш інформативним, оскільки він вказує тільки на зростання сукупної дисперсії сигналу. Активність почала помітно зростати напередодні 2022 року, але для попередніх кризових станів ми не бачимо передвісницької поведінки цього індикатора. Питання передчасної ідентифікації наростання кризового явища найкраще вирішує показник мобільності (Mob). Ми бачимо, що даний показник зростає під час 2015-2016 років, напередодні 2019 року, при настанні коронавірусної пандемії, перед 2023 та 2024 роками. Показник складності Хьорта ($Comp$) реагує асиметричним чином: у той час коли мобільність зростає, показник складності спадає, вказуючи на те, що система прагне до вищого ступеня періодичності або корельованості.

4.3 Висновок

Таким чином, розглянуті інформаційні міри складності дозволяють дослідити певні аспекти складності систем будь-якої природи. Особливо продуктивним являється мультимасштабна версія введених мір. Ретельний аналіз часових рядів для систем різної природи, різного рівня складності, порівняння їх із тестовими сигналами, вивчення поведінки систем у різних (не обов’язково рівноважних, стаціонарних) умовах дозволить зрозуміти природу складності і спрогнозувати можливу поведінку систем у критичних умовах. Так, порівняння вихідного часового ряду з відповідними мірами складності свідчить про очевидне їх реагування на кризові явища. Однак питання використання їх у якості передвісників вимагає додаткових досліджень.

4.4 Завдання для самостійної роботи

1. Дослідіть і порівняйте результати для фінансових рядів, що представляють розвинені компанії (країни, криптовалюти) і такі, що розвиваються. Порівняйте результати. Поясніть, в чому їх схожість та відмінності
2. Яким чином поводять себе міри складності у період фінансових шоків і криз?
3. Наскільки чутливими є результати розрахунків до вибору ширини вікна та кроку?

5. 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

5.1 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 еволюції складних. 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, рекурентного аналізу тощо. 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 широко відомим підходом моделювання особливостей складних систем є обчислення характеристик різних видів 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 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 (https://uk.wikipedia.org/wiki/Ентропія).

Термодинамічна ентропія S, 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 [52]. Він визначив зміну ентропії термодинамічної системи при оборотному процесі як відношення зміни загальної кількості тепла ΔQ до величини абсолютної температури T:

$$\Delta S = \Delta Q / T.$$

Рудольф Клаузіус дав величині S ім'я "*ентропія*", що походить від грецького слова τροπή, "зміна" (зміна, перетворення).

У 1877 році, Людвіг Больцман [53] 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) місце розташування всіх частин обмежене границями судини, (ii) для отримання загальної енергії газу кінетичні енергії атомів підсумовуються. Больцман постулював що

$$S = k_B \ln \Omega,$$

де константу $k_B = 1{,}38 \cdot 10^{-23}$Дж/К ми знаємо тепер як сталу Больцмана, а Ω є числом мікростанів, які можливі в наявному макроскопічному стані. Цей постулат, відомий як принцип Больцмана, може бути оцінений як початок статистичної механіки, що описує термодинамічні системи з використанням статистичної поведінки компонентів. Принцип Больцмана зв'язує мікроскопічні властивості системи (Ω) з однією з її термодинамічних властивостей (S).

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

У випадку дискретних станів квантової механіки кількість станів підраховується звичайним чином. У рамках класичної механіки мікроскопічний стан системи описується координатами q_i й імпульсами p_i окремих частинок, які пробігають неперервні значення. У такому випадку

$$S = k_B \ln \frac{1}{(2\pi\hbar)^s} \int \prod_{i=1}^{s} dq_i dp_i,$$

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

Клод Шеннон [34] запропонував формулу для оцінки невизначеності кодової інформації в каналах зв'язку, звану ентропією Шеннона:

$$S = -k \sum_{i=1}^{n} p_i \ln p_i,$$

де p_i — вірогідність того, що символ i зустрічається в коді, який містить N символів, k — 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 P — апріорна вірогідність деякої події (ймовірність до проведення досвіду), а P_1 — ймовірність цієї події після проведення досвіду. Для простоти вважатимемо, що $P_1 = 1$. За Шенноном, кількість інформації I, яка дає точну відповідь (після проведення експерименту)

$$I = k \log P.$$

Ця кількість інформації, за визначенням, дорівнює одному біту.

Фізичний сенс I — це міра нашого незнання. Іншими словами, I — 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 $P = 1$. Після проведення досвіду ми одержуємо нульову інформацію $I = 0$, оскільки все було відомо до досвіду. Якщо розглядати тіло при кінцевих температурах, то до проведення досвіду число мікростанів, а отже, і P дуже велике. Після проведення досвіду ми одержуємо велику інформацію, оскільки нам стають відомими координати і імпульси всіх частинок.

Аналогія між кількістю інформації і ентропією S, визначуваною з принципу Больцмана, очевидна. Досить покласти множник K рівним постійній Больцмана k_B 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 другий закон термодинаміки до інформації. Нагадаємо, що екстенсивна величина — ця така характеристика системи, яка росте зі збільшенням розмірів системи, тобто, якщо наша система складається з двох *незалежних* підсистем A і B, то ентропію всієї системи можна одержати складанням ентропій підсистем:

$$S(A + B) = S(A) + S(B).$$

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

5.2 Хід роботи

Розглянемо, яким чином ми використовувати ентропійні показники в якості індикаторів або індикаторів-передвісників кризових подій. Перш за все імпортуємо та встановимо необхідні модулі для подальшої роботи:

```
!pip install EntropyHub

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import yfinance as yf
import neurokit2 as nk
import EntropyHub as eh
import warnings
import scienceplots
from tqdm import tqdm

warnings.filterwarnings('ignore')
```

Далі виконаємо налаштування формату виведення рисунків:

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                   # розмір фонтів рисунку
'lines.linewidth': 2,                # товщина ліній
'axes.titlesize': 'small',           # розмір титулки над рисунком
'axes.labelsize': size,              # розмір підписів по осям
'legend.fontsize': size,             # розмір легенди
'xtick.labelsize': size,             # розмір розмітки по осі Ox
'ytick.labelsize': size,             # розмір розмітки по осі Ox
"font.family": "Serif",              # сімейство стилів підписів
"font.serif": ["Times New Roman"],   # стиль підпису
'savefig.dpi': 300,                  # якість збережених зображень
'axes.grid': False                   # побудова сітки на самому рисунку
}

plt.rcParams.update(params)          # оновлення стилю згідно налаштувань
```

У даній роботі виконаємо розрахунки на прикладі одного з найважливіших фондових індексів Японії — Nikkei 225. Індекс обчислюється шляхом визначення простого середнього арифметичного значення цін акцій 225 провідних компаній, які входять до першої секції Токійської фондової біржі. Для отримання значень індексу скористаємось бібліотекою `yfinance`. Будемо розглядати значення до 1-го грудня 2023 року:

```
symbol = '^N225'                        # Символ індексу

end = "2023-12-01"                      # Дата закінчення зчитування даних
data = yf.download(symbol, end=end)     # вивантажуємо дані
time_ser = data['Adj Close'].copy()     # зберігаємо саме ціни закриття

xlabel = 'time, days'                   # підпис по вісі Ox
ylabel = symbol                         # підпис по вісі Oy

np.savetxt(f'{symbol}_initial_time_series.txt', time_ser.values)
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'          # підпис по вісі Ox
ylabel = symbol                    # підпис по вісі Oy
```

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 Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

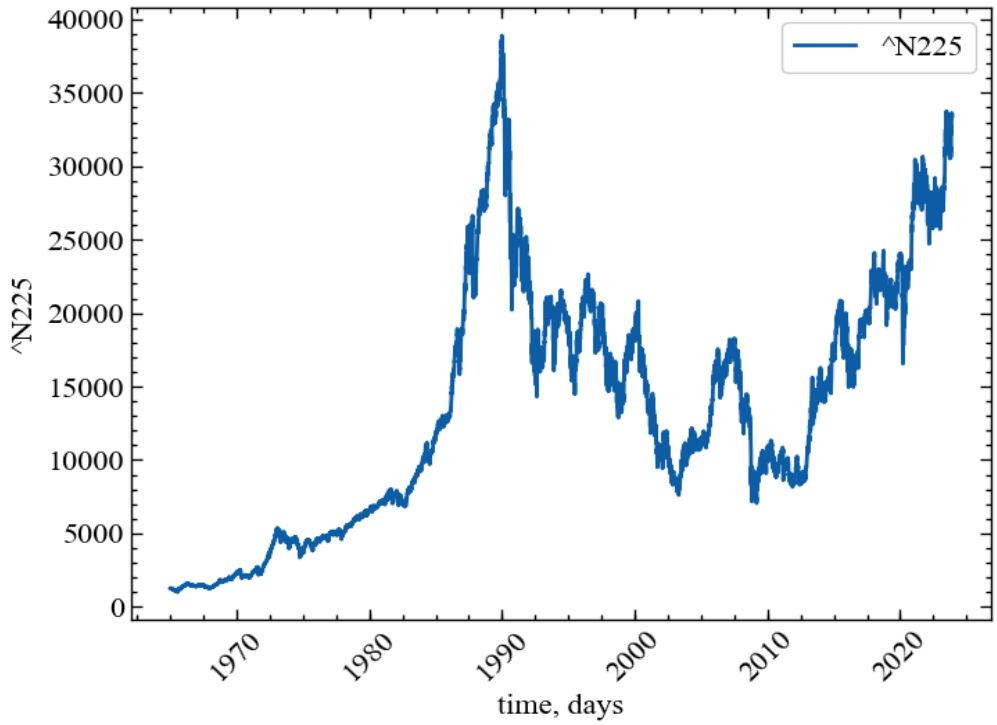


Рис. 5.1: Динаміка щоденних змін фондового індексу N225

Для приведення ряду до стандартизованого вигляду або прибутковостей визначимо функцію `transformations()`:

```
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
```

Для побудови пари часових рядів визначимо функцію `plot_pair()`:

```
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', rotation=35, **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();
```

5.2.1 Апроксимаційна ентропія

Апроксимаційна ентропія (Approximate Entropy, ApEn) є "статистикою регулярності" [54,55], що визначає можливість передбачувати флуктуації в часових рядах [56]. 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 S_N, що складається із N значень $t(1), t(2), t(3), \ldots, t(N)$ вибираються два параметри, d_E та r. Перший з цих параметрів, m, вказує на довжину шаблона, а другий — r — визначає критерій подібності. Досліджуються підпослідовності елементів часового ряду S_N, що складаються з m чисел, взятих, починаючи з номера i, і називаються векторами $p_{d_E}(i)$. Два вектори (шаблони), $p_{d_E}(i)$ та $p_{d_E}(j)$, будуть подібними, якщо всі різниці пар їх відповідних координат є меншими за значення r, тобто якщо

$$|t(i+k) - t(j+k)| < r \quad \text{для} \quad 0 \le k < d_E.$$

Для розглядуваної множини P_{d_E} всіх векторів довжини d_E часового ряду S_N можна розрахувати значення

$$C_{id_E}(r) = n_{id_E}(r)/(N - d_E + 1),$$

де $n_{id_E}(r)$ — кількість векторів у P_{d_E}, що подібні вектору $p_{d_E}(i)$ (враховуючи вибраний критерій подібності r). Значення $C_{id_E}(r)$ є часткою векторів довжини d_E, що мають схожість із вектором такої ж довжини, елементи якого починаються з номера i. Для даного часового ряду обраховуються

значення $C_{id_E}(r)$ для кожного вектора у P_{d_E}, після чого знаходиться середнє значення $C_{d_E}(r)$, що відображає розповсюдженість подібних векторів довжини d_E у ряду S_N. Безпосередньо ентропія подібності для часового ряду S_N з використанням векторів довжини d_E та критерію подібності r визначається за формулою:

$$ApEn(S_N, d_E, r) = \ln C_{d_E}(r) - \ln C_{d_E+1}(r).$$

Тобто, як натуральний логарифм відношення повторюваності векторів довжиною d_E до повторюваності векторів довжиною $d_E + 1$.

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 d_E та r.

```
window = 500      # ширина вікна
tstep = 5         # часовий крок

m = 3             # розмірність вкладень
tau = 1           # часова затримка
r = 0.45          # параметр подібності

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

length = len(time_ser.values)  # довжина самого ряду

ApEn = []                      # масив для зберігання значень ентропії

for i in tqdm(range(0, length-window, tstep)): # фрагменти довжиною window
# з кроком tstep

# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)
```

```
# розраховуємо апроксимаційну ентропію
    Ap, _ = nk.entropy_approximate(signal=fragm,
                                   dimension=m,
                                   delay=tau,
                                   tolerance=r,
                                   corrected=False)
    ApEn.append(Ap)
```

Зберігаємо значення апроксимаційної ентропії до текстового файлу:

```
np.savetxt(f"ApEn_name={symbol}_window={window}_step={tstep}_\
           dim={m}_tau={tau}_radius={r}_sertype={ret_type}.txt", ApEn)
```

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
label_apen = fr'$ApEn$'

file_name_apen = f"ApEn_name={symbol}_window={window}_step={tstep}_\
           dim={m}_tau={tau}_radius={r}_sertype={ret_type}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          ApEn,
          ylabel,
          label_apen,
          xlabel,
          file_name_apen)
```

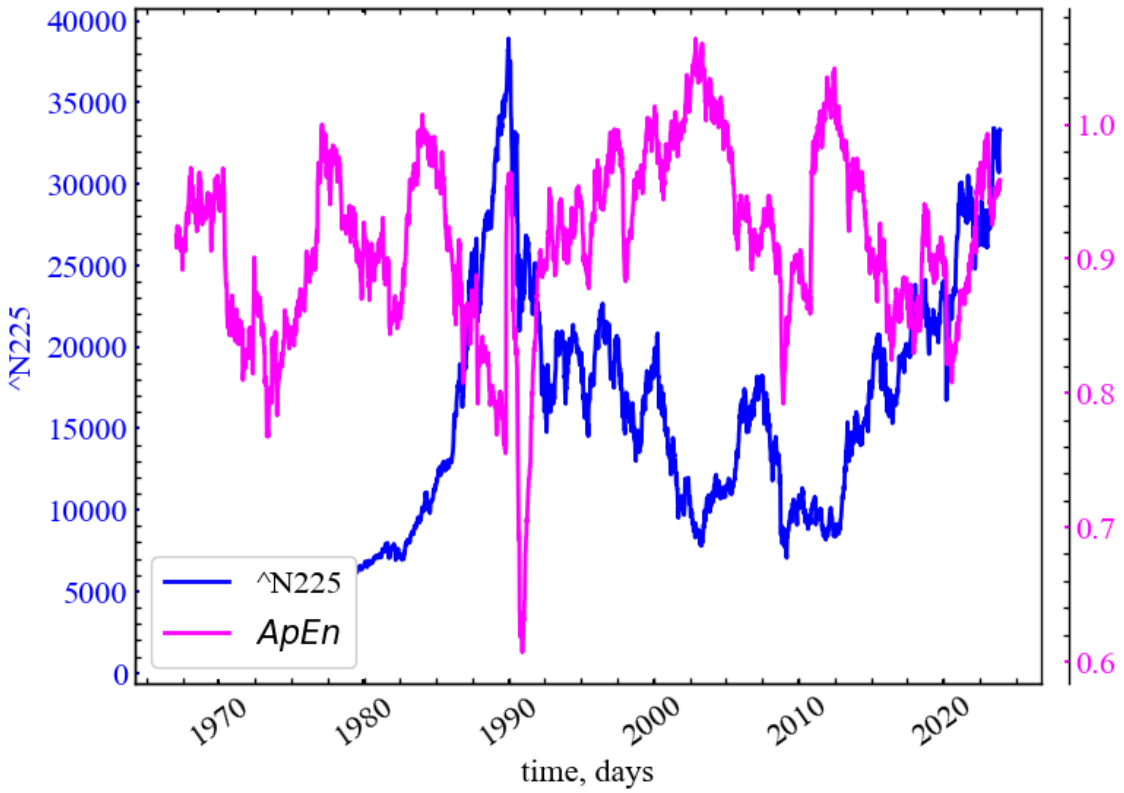


Рис. 5.2: Динаміка фондового індексу N225 та апроксимаційної ентропії

Як можна бачити з представленого рисунку (Рис. 5.2), апроксимаційна ентропія падає у кризові та передкризові моменти часу. Це говорить те, що середнє значення кореляційного інтегралу отриманого для фазового простору розмірністю d_{E+1} не сильно відрізняється від того, що було отримано для фазового простору розмірністю d_E. Тобто, при реконструкції простору в різних вимірах, усі точки просто знаходяться достатньо близько один до одного, не дивлячись на геометричні перетворення атрактора фондового ринку. Це вказує на досить високий ступінь кореляцій цінових флуктуацій індексу N225 і спрямованість трейдерів ринку по одному тренду.

5.2.2 Нечітка ентропія

Однією з модифікацій ентропії Шеннона є **нечітка ентропія** (Fuzzy entropy, FuzzEn) [57–59]. Цей підхід виключає самоподібність між досліджуваними векторами, і замість функції Гевісайда, яка видає або 0, або 1 для схожих векторів, використовується нечітка функція належності, яка у випадку FuzzEn буде асоціювати схожість між двома векторами з реальним значенням у діапазоні $[0, 1]$. Різницю можна побачити на етапі побудови вектора вкладень, де для реконструйованих векторів ми виконуємо детрендування:

$$\vec{X}(i) = \{x(i), x(i+1), \ldots, x(i+d_E-1) - x_0(i)\},\ i = 1, \ldots, N - d_E + 1,$$

$x_0(i) = (d_E)^{-1} \sum_{j=0}^{d_E-1} x\,(i+j)$. Далі, для послідовних вбудованих векторів знаходиться відстань

$$d\left[\vec{X}(i), \vec{X}(j)\right] = \max\left|\vec{X}(i) - \vec{X}(j)\right|, i \geq 1, j \leq N - d_E + 1.$$

У класичній *ApEn* значення відстаней пропускаються через функцію Гевісайда. Нечітка модифікація використовує функції належності для вимірювання належності однієї траєкторії до іншої:

$$D_{i,j} = \mu\left(d\left[\vec{X}(i), \vec{X}(j)\right]\right),$$

$\mu = \exp(-x^{r_2}/r_1)$, а r_1 і r_2 ширина та градієнт експоненціальної функції.

Далі, обчислюється наступна функція, що подібна до кореляційного інтегралу в класичній *ApEn*:

$$\phi^{d_E} = \frac{1}{(N - d_E + 1)} \sum_{i=1}^{N-d_E+1} \left(\frac{1}{N - d_E} \sum_{j=1, j \neq i}^{N-d_E} D_{i,j} \right).$$

Нарешті,

$$FuzzEn\left(\vec{X}, d_E\right) = -[\ln \phi^{d_E+1} - \ln \phi^{d_E}].$$

```
window = 500        # ширина вікна
tstep = 5           # часовий крок

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' - два значення r,
# для gudermannian та linear - 1 значення r,

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

length = len(time_ser.values)  # довжина самого ряду

FuzzEn = [] # масив для зберігання значень ентропії

for i in tqdm(range(0, length-window, tstep)):  # фрагменти довжиною window
# з кроком tstep

# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

# обчислення нечіткої ентропії
    Fuzz, _, _ = eh.FuzzEn(Sig=fragm, m=m, tau=tau, Fx=characteristic_func, r=r)
    FuzzEn.append(Fuzz[-1]) # дожаємо розрахованє значення до масиву значень
```

Зберігаємо значення нечіткої ентропії до текстового файлу:

```
np.savetxt(f"FuzzEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_radius={r}_sertype={ret_type}_\
        memberfunc={characteristic_func}.txt", FuzzEn)
```

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
label_fuzzen = fr'$FuzzEn$'

file_name_fuzzen = f"FuzzEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_radius={r}_sertype={ret_type}_\
        memberfunc={characteristic_func}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          FuzzEn,
          ylabel,
          label_fuzzen,
          xlabel,
          file_name_fuzzen,
          clr='red')
```

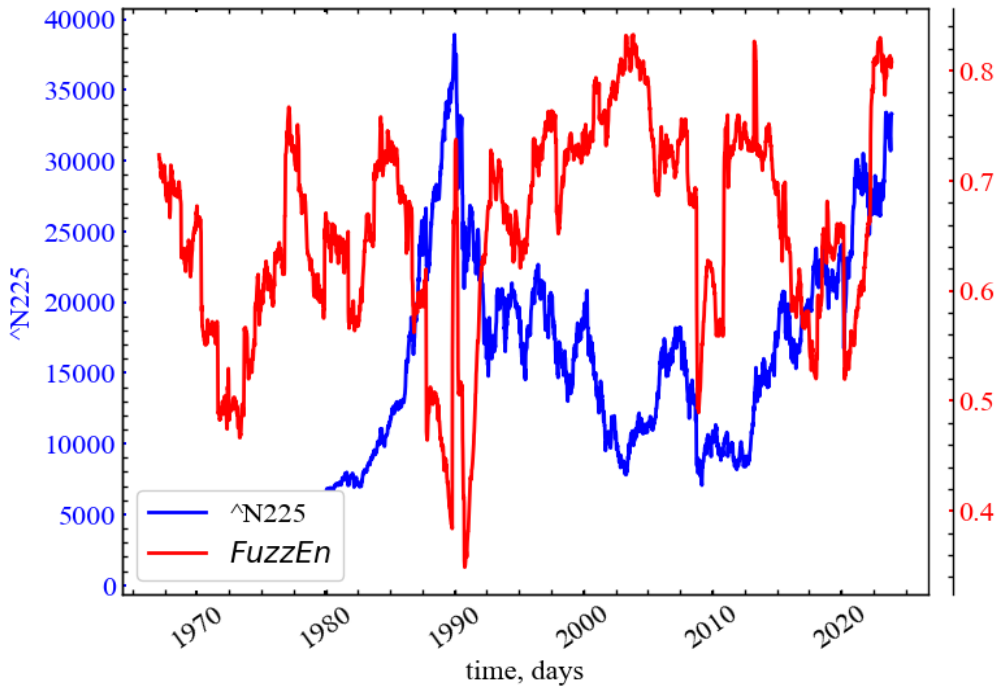


Рис. 5.3: Динаміка фондового індексу N225 та нечіткої ентропії

Рис. 5.3 демонструє спадку динаміку нечіткої ентропії в кризові та передкризові періоди, яка працює по аналогії до апроксимаційної ентропії. Із цього випливає, що нечітка ентропія також вказує на зростання корельованості системи та зростання її трендостійкості через однонаправленність думок трейдерів щодо подальшої динаміки фондового ринку.

5.2.3 Ентропія шаблонів

При розрахунку ApEn беруться до уваги подібності певного вектора $p_n(i)$ до самого себе, що використовується для уникнення можливого значення $\ln 0$ при відсутності подібних до даного векторів. Однак, вказана особливість призводить до нівелювання двох важливих характеристик у ентропії подібності:

- ApEn сильно залежить від довжини розглядуваного шаблона (вектора) і є нижчою, ніж очікується, для векторів малої розмірності;
- ApEn не враховує відносну щільність даних.

Це означає, що коли значення ApEn для одного ряду є більшим, ніж для іншого, то воно повинно залишатись таким (проте не є) для будь-яких можливих початкових умов. Такий висновок тим більш важливий, оскільки ApEn рекомендується в якості міри порівняння двох наборів даних різними авторами.

Враховуючи вказані обмеження, розроблена для розрахунку інша характеристика, — **ентропія шаблонів** (Sample Entropy, SampEn) [60].

При розрахунку SampEn, на відміну від алгоритму ApEn, додаються дві умови:

- не враховується подібність вектора самому собі;
- при розрахунку значень умовних ймовірностей SampEn не використовується довжина векторів.

На основі аналізу вищезазначеного можна зробити висновок про те, що SampEn:

- більше, ніж ApEn, відповідає теорії випадкових чисел для ряду із відомою функцією щільності розподілу;
- зберігає відносну щільність, в той час як ApEn втрачає дану характеристику;
- додає значно меншу помилку до розрахованого значення у випадку використання векторів малої розмірності.

```
window = 500            # ширина вікна
tstep = 5               # часовий крок

m = 3                   # розмірність вкладень
tau = 1                 # часова затримка
r = 0.4                 # параметр подібності

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

length = len(time_ser.values)  # довжина самого ряду

SampEn = []     # масив для зберігання значень ентропії шаблонів

for i in tqdm(range(0, length-window, tstep)):  # фрагменти довжиною window
# з кроком tstep

# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

# обчислення ентропії шаблонів
    Samp, _ = nk.entropy_sample(signal=fragm,
                                dimension=m,
                                delay=tau,
                                tolerance=r)
    SampEn.append(Samp)
```

Зберігаємо значення ентропії шаблонів до текстового файлу:

```
np.savetxt(f"SampEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_radius={r}_sertype={ret_type}.txt", SampEn)
```

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
label_sampen = fr'$SampEn$'

file_name_sampen = f"SampEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_radius={r}_sertype={ret_type}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          SampEn,
          ylabel,
          label_sampen,
          xlabel,
          file_name_sampen,
          clr='darkgreen')
```

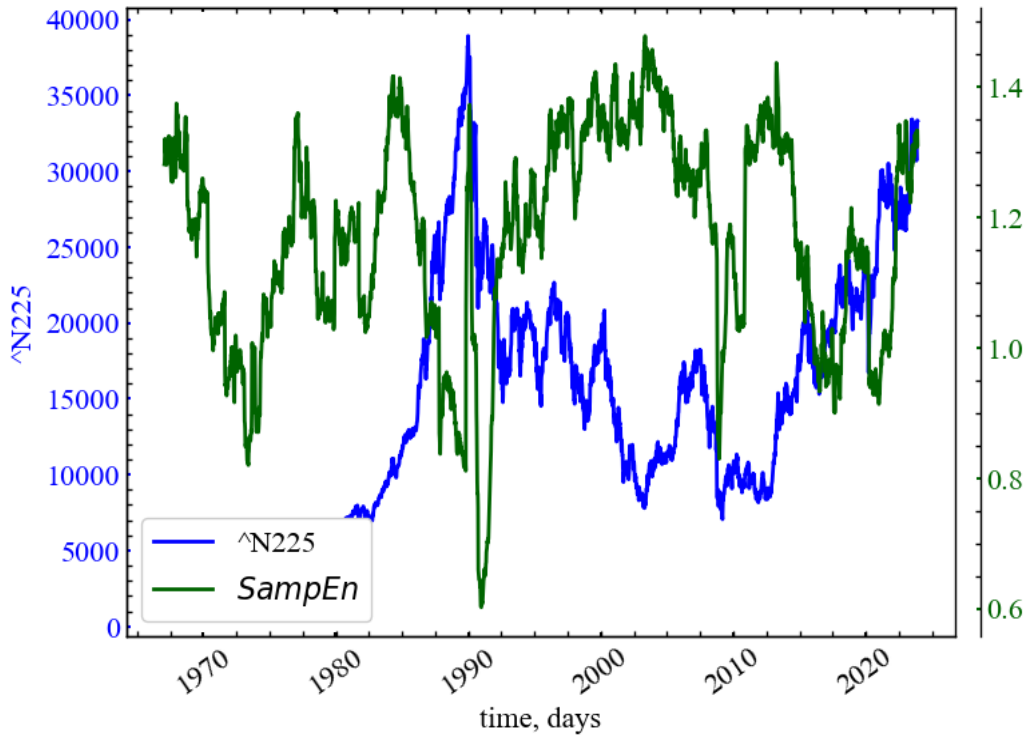


Рис. 5.4: Динаміка фондового індексу N225 та ентропії шаблонів

Рис. 5.4 демонструє, що ентропія шаблонів спадає в передкризові періоди фондового ринку, що вказує на зростання корельованості траєкторій реконструйованого фазового простору індексу N225. Це говорить про те, що ринок у передкрахові періоди стає більш упорядкованим і трендостійким.

5.2.4 Ентропія перестановок

Ентропія перестановок (Permutation entropy, PEn) — це міра з теорії хаосу, запропонована Бандтом і Помпе [61], і характеризується концептуальною простотою і швидкістю обчислень. Ідея PEn базується на звичайній ентропії Шеннона, але використовує патерни перестановок — порядкові відношення між значеннями системи. Порівняно з іншими мірами складності має певні переваги, такі як стійкість до шуму та інваріантність до нелінійних монотонних перетворень [62].

Як і в попередніх типах ентропії, ми реконструюємо часовий ряд із N значень з фіксованою розмірністю вбудовування d_E та часовою затримкою τ, за вкладеною матрицею формуємо часові векторні послідовності

$$\vec{X}(i) = \{x(i), x(i+1), \ldots, x(i+[d_E-1]\tau)\},$$

і в результаті отримуємо $N - [d_E - 1]\tau$ векторів.

Кожен елемент $\vec{X}(i)$ перетворюється в числові ранги відповідно до їх порядку. Наприклад, для $d_E = 2$ і $\tau = 1$ та часового ряду $\vec{X}(i) = (-0.1, 0.4, 3.2, 12.0, 6.5)$, вбудована матриця матиме такі пари: $\vec{X}(1) = -0.1{,}0.4$, $\vec{X}(2) = 0.4{,}3.2$, $\vec{X}(3) = 3.2{,}12.0$, $\vec{X}(4) = 12.0{,}6.5$.

Далі ми формуємо порядкові послідовності відповідно до їх числового порядку. Такі вектори як $\vec{X}(1), \vec{X}(2), \vec{X}(3)$ задовольняють умові $x(i) < x(i+1)$ і один вектор $\vec{X}(4)$ задовольняє умові $x(i) > x(i+1)$. Можна розглянути $d_E!$ можливих перестановок порядку d_E. У нашому прикладі є лише 2! шаблони: $\pi_1 = 0{,}1, \pi_2 = 1{,}0$.

Для кожного шаблону ми визначаємо його відносну частоту:

$$p(\pi) = \#\{\vec{X}(i) \text{ має шаблон } \pi\}/(N - [d_E - 1]\tau).$$

Імовірність знаходження вектора по шаблону π_1 дорівнює $3/4$ і по шаблону π_2 дорівнює $1/4$, тобто, ми формуємо розподіл імовірностей $P = \{p(\pi_1), \ldots, p(\pi_{d_E})\}$. Нарешті, даний вид ентропії може бути розрахований у той самий спосіб, що й ентрпія Шеннона:

$$PEn(\vec{X}, d_E) = -\sum_{i=1}^{d_E} p\,(\pi_i) \ln p(\pi_i).$$

Для зручності PEn нормалізується згідно наступного рівняння [63]:

$$\overline{PEn(\vec{X}, d_E)} = PEn(\vec{X}, d_E)/PEn_{max},$$

де $PEn_{max} = \ln D!$, а нормалізована ентропія перестановок знаходиться в діапазоні $0 \le \overline{PEn(\vec{X}, d_E)} \le 1$.

```
window = 500          # ширина вікна
tstep = 5             # часовий крок

m = 4                 # розмірність вкладень
tau = 3               # часова затримка

Type = 'none'         # none - класична
                      # finegrain - Дрібнозерниста ентропія перестановок
                      # modified - Модифікована ентропія перестановок
                      # weighted - Зважена ентропія перестановок
                      # ampaware - Амплітудно-орієнтована ентропія перестановок
                      # edge - Ентропія перестановки граней
                      # uniquant - Рівномірна ентропія перестановок на основі
квантування
```

```
tpx = -1    # finegrain tpx - параметр α, додатний скаляр (за замовчуванням: 1)
            # ampaware tpx - параметр A, значення в діапазоні [0, 1] (за
замовчуванням: 0.5)
            # edge tpx - параметр чутливості r, скаляр > 0 (за замовчуванням: 1)
            # uniquant tpx - параметр L, ціле число > 1 (за замовчуванням: 4)

log = np.exp(1)   # основа логарифма
norm = True       # нормування ентропії

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 самого ряду

PEn = []        # масив для зберігання значень нормалізованої перм. ентропії

for i in tqdm(range(0, length-window, tstep)):  # фрагменти довжиною window
# з кроком tstep

# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

# обчислюємо ентропію перестановок
    _, Pnorm, cPE = eh.PermEn(fragm,
                              m=m,
                              tau=tau,
                              Typex=Type,
                              tpx=tpx,
                              Logx=log,
                              Norm=norm)

    PEn.append(Pnorm[-1])
```

Зберігаємо значення пермутаційної ентропії до текстового файлу:

```
np.savetxt(f"PEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_sertype={ret_type}_type={Type}_param={tpx}.txt", PEn)
```

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
label_permen = fr'$PEn$'

file_name_perm = f"PEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_sertype={ret_type}_type={Type}_param={tpx}"
```

Візуалізуємо результат для ентропії перестановок:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
```

```
            PEn,
            ylabel,
            label_permen,
            xlabel,
            file_name_perm,
            clr='indigo')
```

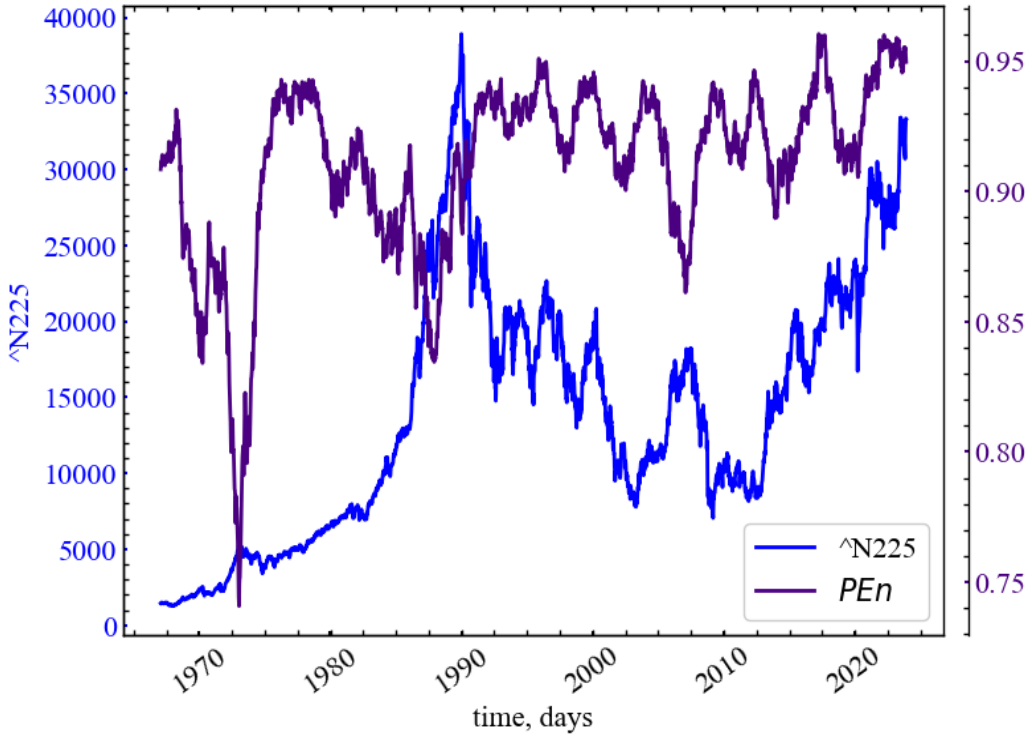


Рис. 5.5: Динаміка фондового індексу N225 та ентропії перестановок

На представленому рисунку (Рис. 5.5) видно, що PEn спадає в кризові та передкризові періоди на фондовому ринку. Це вказує на зростання ймовірності появи одного конкретного патерна для подальшої динаміки ринку, а отже й кількості очікуваної інформації при аналізі флуктуацій індексу N225.

5.2.5 Ентропія сингулярного розкладу

Ентропію сингулярного розкладу (Singlular value decomposition entropy, SVDEn) [64] можна інтуїтивно розглядати як показник того, скільки власних векторів потрібно для адекватного пояснення набору даних. Іншими словами, вона вимірює багатство ознак: чим вища SVDEn, тим більше ортогональних векторів потрібно для адекватного пояснення стану простору. Подібно до інформаційого показника Фішера, SVDEn базується на розкладанні сингулярного значення реконструйованого методом часових затримок сигналу.

```
window = 500  # ширина вікна
tstep = 5     # часовий крок

m = 3         # розмірність вкладень
```

```
tau = 1      # часова затримка

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

length = len(time_ser.values)  # довжина самого ряду

SVDEn = [] # масив для зберігання значень розподільної ентропії

for i in tqdm(range(0, length-window, tstep)):  # фрагменти довжиною window
# з кроком tstep

# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

# обчислення розподільної ентропії
    svden, _ = nk.entropy_svd(signal=fragm,
                              dimension=m,
                              delay=tau)

    SVDEn.append(svden)
```

Зберігаємо значення розподільної ентропії до текстового файлу:

```
np.savetxt(f"SVDEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_sertype={ret_type}.txt", SVDEn)
```

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
label_svden = fr'$SVDEn$'

file_name_svden = f"SVDEn_name={symbol}_window={window}_step={tstep}_\
        dim={m}_tau={tau}_sertype={ret_type}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          SVDEn,
          ylabel,
          label_svden,
          xlabel,
          file_name_svden,
          clr='darkorange')
```

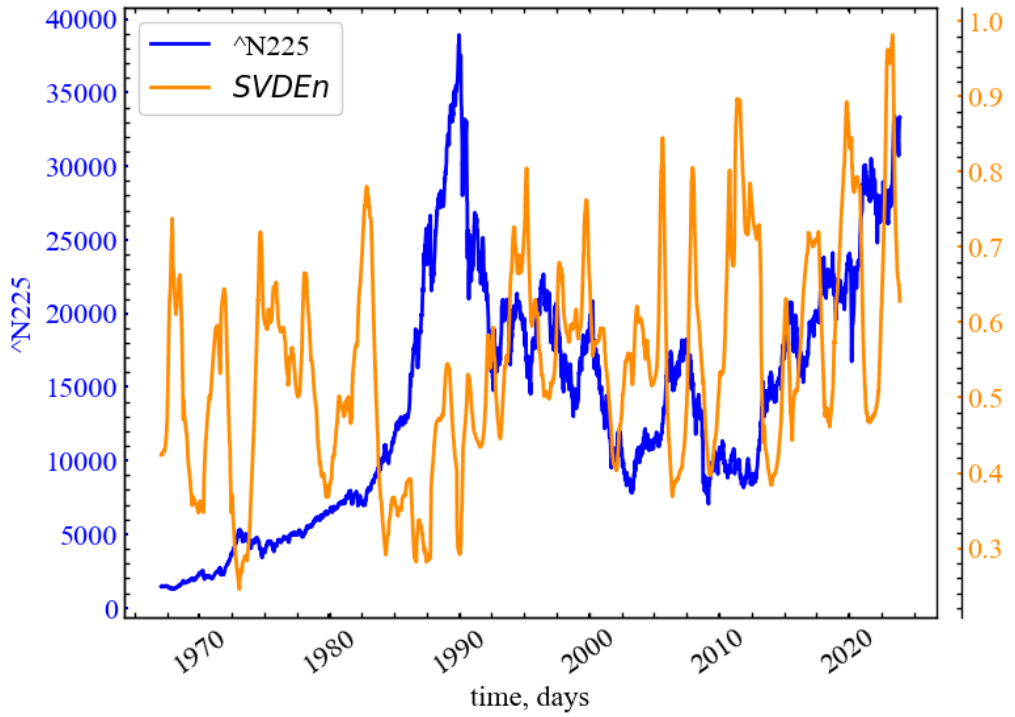


Рис. 5.6: Динаміка фондового індексу N225 та ентропії сингулярного розкладу

Рис. 5.6 показує, що ентропія сингулярного розкладу спадає у (перед)кризові періоди, що говорить про зростання кореляцій на фондовому ринку. Оскільки SVDEn базується на розподілі власних векторів, можна зробити припущення, що в передкризові моменти часу динамікою ринку керують один або декілька власних векторів, які і є рушійною складновою досліджуваного індексу.

5.2.6 Дисперсійна ентропія

Для заданого одновимірного сигналу довжини N: $x = \{x_1, x_2, \dots, x_N\}$, алгоритм **дисперсійної ентропії** (Dispersion entropy, DispEn) включає 4 основних кроки [65]:

1) Спочатку x_j $(j = 1,2,\dots N)$ відображаються на c класів, позначених від 1 до c. Для цього існує ряд лінійних та нелінійних підходів. Хоча лінійний алгоритм відображення є найшвидшим, коли максимальні та/або мінімальні значення часового ряду набагато більші або менші за середнє/медіанне значення сигналу, більшість значень x_i віднесено лише до кількох класів. Таким чином, ми спочатку використовуємо **нормальну кумулятивну функцію розподілу** (normal cumulative distribution function, NCDF) для відображення x в $y = \{y_1, y_2, \dots, y_N\}$ від 0 до 1. Далі виконується лінійний алгоритм присвоєння кожному y_j цілого числа від 1

до c. Для цього для кожного члена відображеного сигналу ми використовуємо $z_j^c = \text{round}(y_j + 0.5)$, де z_j^c показує j-й член класифікованого часового ряду, а округлення передбачає або збільшення, або зменшення числа до наступної цифри. Варто зазначити, що цей крок можна виконати за допомогою інших лінійних та нелінійних методів відображення.

2) Кожен вектор $\mathbf{z}_i^{d_E,c}$ з розмірністю d_E та часовою затримкою τ має вид $\mathbf{z}_i^{d_E,c} = \{z_i^c, z_{i+\tau}^c, \dots, z_{i+(d_E-1)\tau}^c\}$, $i = 1, \dots, N - (d_E - 1)\tau$ і проектується на дисперсійний шаблон $\pi_{v_0,v_1,\dots,v_{d_E-1}}$, де $z_i^c = v_0, z_{i+\tau}^c = v_1, \dots, z_{i+(d_E-1)\tau}^c = v_{d_E-1}$. Кількість можливих дисперсійних шаблонів, що може бути присвоєна кожному вектору $\mathbf{z}_i^{d_E,c}$, дорівнює c^m, оскільки сигнал має d_E елементів і кожному елементу може бути присвоєно ціле значення від 1 до c.

3) Для всіх потенційних c^m дисперсійних шаблонів розраховується відносна частота:

$$p\left(\pi_{v_0,v_1,\dots,v_{d_E-1}}\right) = \frac{\#\left\{i | i \le N - (d_E - 1)\tau, \mathbf{z}_i^{d_E,c} \text{ має шаблон } \pi_{v_0,v_1,\dots,v_{d_E-1}}\right\}}{N - (d_E - 1)\tau}.$$

4) Нарешті, опираючись на формулу ентропії Шеннона, DispEn розраховується як

$$DispEn(x, d_E, c, \tau) = -\sum_{\pi=1}^{c^{d_E}} p\left(\pi_{v_0,v_1,\dots,v_{d_E-1}}\right) \ln p\left(\pi_{v_0,v_1,\dots,v_{d_E-1}}\right).$$

Коли всі можливі дисперсійні шаблони мають однакову ймовірність, отримуємо найбільше значення DispEn, яке має величину $\ln c^{d_E}$. І навпаки, якщо тільки одне $p(\pi_{v_0,v_1,\dots,v_{d_E-1}})$ відрізняється від нуля (абсолютно регулярний/передбачуваний сигнал), отримуємо найменше значення DispEn.

```
window = 500 # ширина вікна
tstep = 5    # часовий крок

fluct = False # флуктуаційно-дисперсійна ентропія
m = 3 # розмірність вкладень
tau = 1 # часова затримка
rho = 1 # параметр для Type="finesort", позитивний скаляр (за замовчуванням 1)
classes = 6 # кількість символів, що задіяні при перетворені

type = 'ncdf' # тип символьного перетворення ряду:
```

```
# "ncdf" - Нормалізована кумулятивна функція розподілу
# "kmeans" - Алгоритм кластеризації K-середніх
# "linear" - Лінійна сегментація діапазону сигналу
# "finesort" - Ентропія дрібнодисперсного розсіювання

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

length = len(time_ser.values)  # довжина самого ряду

DispEn = [] # масив значень для зберігання дисперсійної ентропії

for i in tqdm(range(0, length-window, tstep)):  # фрагменти довжиною window
# з кроком tstep

# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

# обчислюємо дисперсійну ентропію
    Disp, _ = nk.entropy_dispersion(signal=fragm,
                                    dimension=m,
                                    delay=tau,
                                    c=classes,
                                    symbolize=type,
                                    fluctuation=fluct,
                                    rho=rho)
    DispEn.append(Disp)
```

Зберігаємо значення дисперсійної ентропії до текстового файлу:

```
np.savetxt(f"DispEn_symbol={symbol}_window={window}_step={tstep}_d_e={m}_tau={ta
u}_\
           series_type={ret_type}_fluct={fluct}_rho={rho}_\
           classes={classes}_type={Type}.txt", DispEn)
```

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
label_dispen = fr'$DispEn$'

file_name_dispen =
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}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          DispEn,
```

```
            ylabel,
            label_dispen,
            xlabel,
            file_name_dispen,
            clr='coral')
```

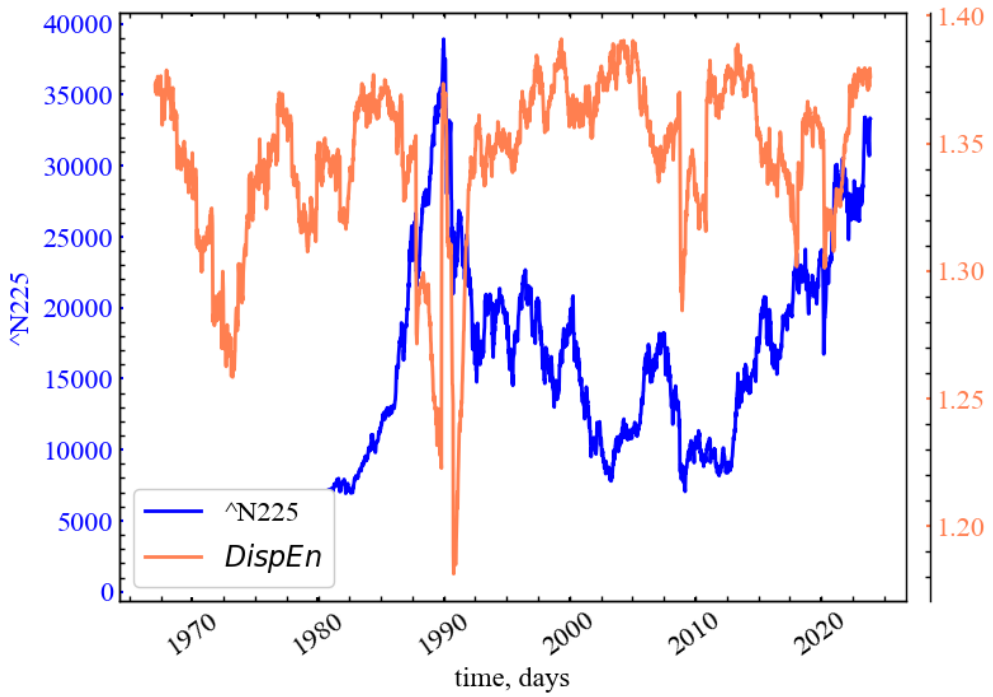


Рис. 5.7: Динаміка фондового індексу N225 та дисперсійної ентропії

Рис. 5.7 демонструє, що дисперсійна ентропія падає напередодні крахових періодів. Особливо помітним це предстає для крахів 1970, 1990, 2010 та 2020. Це говорить про те, що розподіл дисперсійних шаблонів стає зміщенним, що й відзеркалюється у спаді ентропії. Це також вказує на періодизацію ринку. Для періодів, коли дисперсійна ентропія максимальна, очікування трейдерів також залишаються різносторонніми, що робить ринок більш непередбачуваним.

5.2.7 Спектральна ентропія

Спектральна ентропія (Spectral entropy, SE або SpEn) [66] розглядає нормовану **щільність спектра потужності** (power spectral density, PSD) сигналу в частотній області як розподіл імовірностей і обчислює його ентропію Шеннона:

$$SpEn = -\sum P(f) \log P(f).$$

Сигнал з однією частотною складовою (наприклад, чиста синусоїда) має найменшу ентропію. З іншого боку, сигнал з усіма частотними компонентами однакової потужності (білий шум) дає найбільшу ентропію.

```
window = 500 # ширина вікна
tstep = 5    # часовий крок

num_bins = 30 # якщо передано ціле число, розділить PSD
# на декілька частотних діапазонів

method = 'fft' # метод для розрахунку щільності спектра потужності:
               # welch
               # fft
               # multitapers
               # lombscargle
               # burg

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

length = len(time_ser.values)  # довжина самого ряду

SpEn = [] # масив значень для зберігання спектральної ентропії

for i in tqdm(range(0, length-window, tstep)):  # фрагменти довжиною window
# з кроком tstep

# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

# обчислюємо спектральну ентропію
    spec, _ = nk.entropy_spectral(signal=fragm,
                                  bins=num_bins,
                                  method=method)

    SpEn.append(spec)
```

Зберігаємо значення спектральної ентропії до текстового файлу:

```
np.savetxt(f"SpEn_symbol={symbol}_window={window}_step={tstep}_\
           series_type={ret_type}_bins={num_bins}_psd={method}.txt", SpEn)
```

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
label_spen = fr'$SpEn$'

file_name_spen = f"SpEn_symbol={symbol}_window={window}_step={tstep}_\
                   series_type={ret_type}_bins={num_bins}_psd={method}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          SpEn,
          ylabel,
          label_spen,
          xlabel,
          file_name_spen,
          clr='deeppink')
```

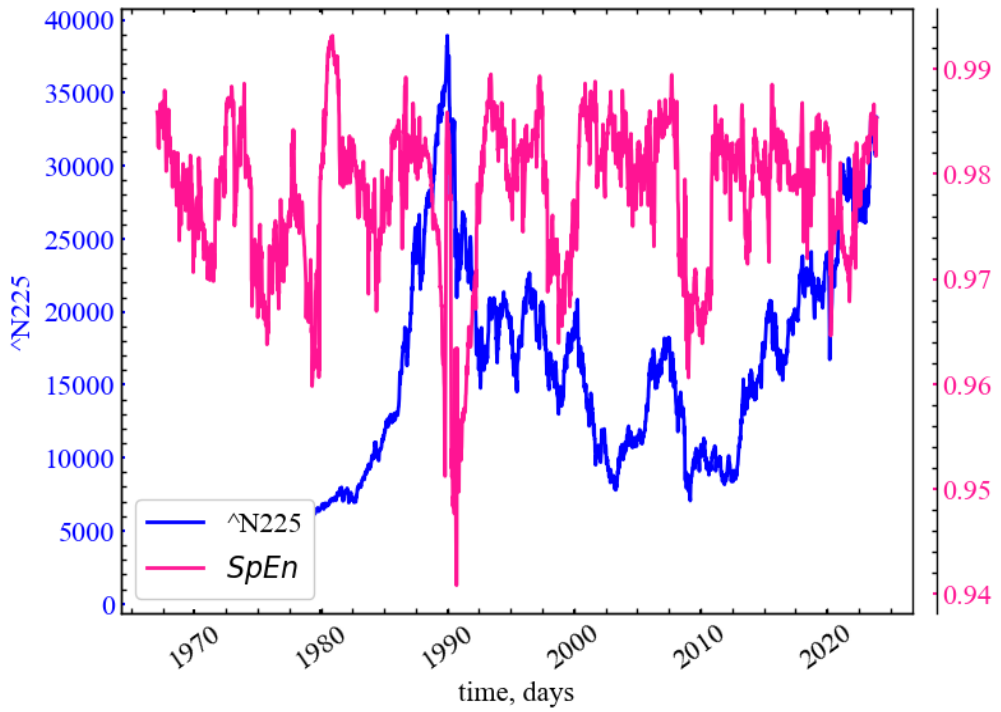


Рис. 5.8: Динаміка фондового індексу N225 та спектральної ентропії

З Рис. 5.8 видно, що спектральна ентропія починає падати в передкризовий період, що говорить про зміщення спектра потужності в конкретну область частот. Це вказує на періодизацію динаміки ринку, що у свою чергу вказує на зростання кореляцій і трендостійкості ринку.

5.3 Висновок

На прикладі ентропійних мір складності перевірено гіпотезу про зв'язок мір складності та кризових явищ, висунуту на основі теорії складних систем. У рамках алгоритму ковзного вікна за набором ентропійних показників було показано, що фінансові крахи характеризуються зміною складності: у передкризовий період, як правило, ми можемо спостерігати упорядкування системи, а в кризовий та післякризовий періоди зростання хаотичності. Порівняння ентропійних характеристик відкриває можливість передчасної

ідентифікації та попередження ня кризових явищ у системах різної природи та складності.

Таким чином, представлені індикатори-передвісники кризових явищ, теоретично, дозволяють обійти потребу в значних обчислювальних ресурсах і досить дискусійних методів прогнозування цінових коливань і їх трендів.

5.4 Завдання для самостійної роботи

1. Проведіть порівняльний аналіз ентропійних показників для тестових часових рядів
2. Побудуйте часові ряди, які у визначених точках характеризуються кризами і покажіть, чи є (і які саме) ентропійні міри індикаторами, або передвісниками кризових явищ
3. Зробіть загальні висновки і оформіть звіт

6. Лабораторна робота № 6

Тема. Фрактальний аналіз складних систем

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

6.1 Теоретичні відомості

6.1.1 Визначення фрактала

Фракталами (fractals) називають геометричні об’єкти: лінії, поверхні, просторові тіла, що мають сильно шорстку поверхню або форму і характеризуються властивістю *самоподібності* [67–69]. Слово фрактал походить від латинського слова *fractus* і перекладається як дробовий, ламаний. Самоподібність як основна характеристика фрактала означає, що він більш-менш однорідно змінюється в широкому діапазоні масштабів. Так, при збільшенні маленькі фрагменти фрактала стають 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 l_{min}, такий, що на масштабі $l \approx l_{min}$ властивість самоподібності зникає. Окрім цього, на достатньо великих масштабах довжин $l > l_{max}$, де l_{max} — характерний геометричний розмір об’єктів, ця властивість самоподібності також порушується. Тому властивості природніх фракталів розглядаються лише на масштабах l, що задовільняють відношення $l_{min} \ll l \ll l_{max}$. 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 частинки, а також скінченність 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.

6.1.2 Довжина берегової лінії

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 [70–73]. Нехай, наприклад, відстань по прямій між розсташованими на береговій лінії точками A та B дорівнює R (див. Рис. 6.1).

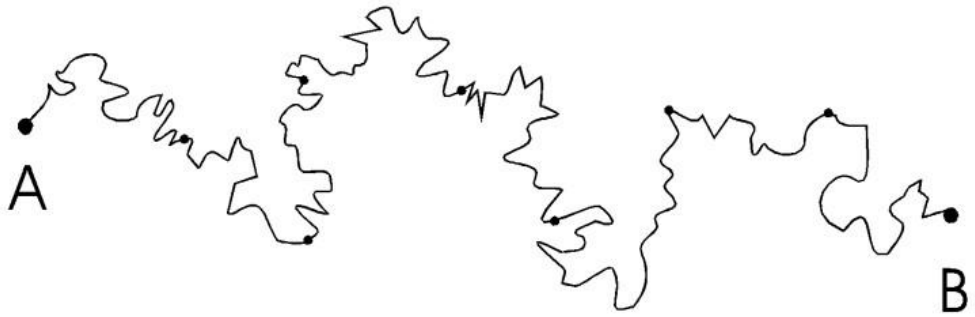


Рис. 6.1: Визначення довжини берегової лінії між точками A та B [74]

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, $l = 10$ км. Довжину берегової лінії в кілометрах між точками A і B 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 $l = 1$ км.

Виявляється, що результат цих вимірювань буде різним. При зменшенні масштабу l 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 l величина L — довжина берегової лінії — не прагне до кінцевого межі, а збільшується за степеневим законом

$$L \approx l(R/l)^D, \tag{6.1}$$

а $D > 1$ — деякий степеневий показник, котрий іменується **фрактальною розмірністю** (fractal dimension) берегової лінії [73]. Чим більше значення D, тим більш ломаною або деталізованою представляється ця берегова лінія. Походження залежності (6.1) 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 L.

Таким чином, ми бачимо, що для визначення довжини берегової лінії L за допомогою жорсткого масштабу l, необхідно зробити $N = L/l$ кроків, причому величина L змінюється з l так, що N залежить від l за законом $N \approx (R/l)^D$. 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 L, яка апроксимує, за наближення до нуля довжини її ланки l є скінченною. У результаті для гладкої кривої її фрактальна розмірність $D = 1$, тобто збігається з топологічною.

6.1.3 Фрактальна розмірність множин

Вище було введено поняття про фрактальну розмірність берегової лінії. Дамо тепер загальне визначення цієї величини. Нехай d — звичайна Евклідова розмірність простору, в якому розташований наш фрактальний об'єкт ($d = 1$ — лінія, $d = 2$ — площина, $d = 3$ — звичайний тривимірний простір). Покриємо тепер цей об'єкт цілком d-мірними "кулями" радіуса 1. Припустимо, що нам потребувалося для цього не менше, ніж $N(l)$ куль. Тоді, якщо за досить малих l величина $N(l)$ змінюється з l за степеневим законом

$$N(l) \sim 1/l^D, \tag{6.2}$$

тоді D — називається **хаусдорфовою** (Hausdorff) або **фрактальною розмірністю** цього об'єкта [75]. Очевидно, що ця формула еквівалентна відношеню $N \approx (R/l)^D$, що використовувалось вище для визначення довжини берегової лінії. Рівняння (6.3) можна переписати у вигляді

$$D = -\lim_{l \to 0} \ln N(l) / \ln l. \qquad (6.3)$$

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

6.1.4 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 Мінковського [76]. Розмірність Хаусдорфа є найпростішою фрактальною розмірністю. Але її обчислювальна складність є високою, що ускладнює її практичне застосування. Розмірність Мінковського є відносно простою, і фрактальну розмірність сигналу можна отримати, регулюючи розмір довжини сторони клітини в межах якої визначається "шорсткість" поверхні сигналу. Тому вона є широко впізнаваємою та застосовуваною. Який з показників фрактальної розмірності найточніше описує складність сигналу та здатний ідентифікувати кризові явища і представляє ключовий момент цієї лабораторної роботи.

6.1.4.1 R/S-аналіз

Метод *R/S*-аналізу (Rescaled range, *R/S* analysis), розроблений Мандельбротом та Уоллесом [77], базується на попередньо створеному методі гідрологічного аналізу Херста [78,79], і дозволяє обчислювати параметр самоподібності H, який вимірює інтенсивність довготривалих залежностей у часовому ряді. Коефіцієнт H, 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, $G(t) \equiv \ln x(t + \Delta t) - \ln x(t)$, де $x(t)$ — значення вихідного часового ряду в момент t, Δt — часовий крок. Отримана послідовність $G(t)$ розбивається на d підпослідовностей довжини n.

Для кожної підпослідовності $m = 1, \ldots, d$:

1. Шукається середнє значення μ_m та стандартне відхилення S_m.
2. Дані нормалізуються шляхом віднімання середнього значення послідовності $X_{i,m} = G_{i,m} - \mu_m$, $i = 1, \ldots, n$.
3. Знаходиться кумулятивна сума послідовності Xів: $Y_{i,m} = \sum_{j=1}^{i} X_{j,m}$, $i = 1, \ldots, n$.
4. У межах кожної підпослідовності знаходиться розмах між максимальним та мінімальним значеннями: $R_m = \max\{Y_{1,m}, \ldots, Y_{n,m}\} - \min\{Y_{1,m}, \ldots, Y_{n,m}\}$, який стандартизується середнім квадратичним відхиленням R_m/S_m.
5. Обчислюється середнє $(R/S)_n$ нормованих значень розмаху для всіх підпослідовностей довжини n.

R/S-статистика, обрахована таким чином, відповідає співвідношенню $(R/S)_n \cong cn^H$, де значення H може бути отримане шляхом обчислення $(R/S)_n$ для послідовностей інтервалів зі збільшенням часового горизонту:

$$\log(R/S)_n = \log c + H \log n\,. \tag{6.4}$$

Знайти коефіцієнт Херста можна, побудувавши залежність $(R/S)_n$ vs. n у подвійному логарифмічному масштабі і взявши коефіцієнт нахилу прямої, яка інтерполює точки отриманого графіка. Якщо значення $H = 0.5$, говорять про послідовність, що представляє собою **білий шум**; $0.5 < H \leq 1$ свідчить про **персистентний** (трендостійкий) ряд, коли існує тенденція слідування великих значень ряду за великими і навпаки; $H < 0.5$ вказує на **антиперсистентний** (mean-reversion) ряд.

При збільшенні часового горизонту коефіцієнт нахилу інтерполюючої прямої повинен прямувати до значення $H = 0.5$; 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

$$D_f = 2 - H.$$

Якщо для берегової лінії ми визначали масштабування її довжини L в залежності від зміни l, то у випадку з *R/S*-аналізом ми визначаємо зміну нормованого розмаху значень ряду в межах масштабу n

6.1.4.2 Аналіз детрендованих флуктуацій

Аналіз детрендований флуктуацій (Detrended fluctuation analysis, DFA) [80] 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 DFA 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 [81].

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

Розглянемо DFA найнижчого порядку.

1. Для часового ряду довжини N знаходиться кумулятивна сума, $y(k) = \sum_{i=1}^{k}(x_i - \bar{x})$, де x_i — це i-те значення часового ряду, $\bar{x}$ — його середнє значення, $k = 1, \ldots, N$.
2. Отриманий ряд $y(k)$ розбивається на m підпослідовностей (вікон) однакової ширини n і для кожної підпослідовності (у кожному вікні) виконується наступне:
   - за допомогою методу найменших квадратів знаходиться локальний лінійний тренд $y_t(k)$;
   - підпослідовність детрендується шляхом віднімання значення локального тренду $y_t(k)$ від значень ряду $y(k)$, що належать послідовності t;
   - знаходиться середнє $\bar{y_t}$ детрендований значень.

Для отриманих таким чином значень на всіх підпослідовностях знаходиться:

$$F_n = \sqrt{\bar{y_t} \cdot m^{-1}},$$

де n — кількість точок у підпослідовності (ширина вікна), m — кількість підпослідовностей, $\bar{y_t}$ — середнє детрендованих значень для підпослідовності t.

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

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

- для випадкового ряду (перемішаного чи "сурогатного") $\alpha = 0.5$;
- при наявності лише короткочасових кореляцій α може відрізнятись від 0.5, проте має тенденцію прямувати до 0.5 при збільшенні розміру вікна;
- Значення $0.5 < \alpha \leq 1.0$ показує персистентні довгочасові кореляції, що відповідають степеневому закону;
- $0 < \alpha < 0.5$ означає антиперсистентний ряд;
- спеціальний випадок, коли $\alpha = 1$, означає наявність $1/f$ шуму.
- для випадків, коли $\alpha \geq 1$, кореляції існують, проте перестають відображувати степеневу залежність;
- випадок $\alpha = 1.5$ свідчить про броунівський шум, який представляє інтегрований білий шум.

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

$$C(L) \sim L^{-\gamma}.$$

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

$$P(f) \sim f^{-\beta}. \tag{6.5}$$

Відповідні показники виражаються через наступні відношення:

- $\gamma = 2 - 2\alpha$;
- $\beta = 2\alpha - 1$.

У DFA другого порядку (DFA2) обчислюються відхилення $F^2(v, s)$ профілю від інтерполяційного многочлена другого порядку. Таким чином, вилучаються впливи можливих лінійних та параболічних трендів для масштабів,

більших за розглядувані. Взагалі, у DFA порядку n обчислюються відхилення профілю від інтерполяційного многочлена n-го порядку, що вилучає вплив всіх можливих трендів порядків до $(n-1)$ для масштабів, більших від розміру вікна.

Потім обчислюється найближчий поліном $y_v(s)$ для профілю на кожному із $2N_s$ сегментів v і визначається відхилення

$$F^2(v,s) \equiv \frac{1}{s}\sum_{i=1}^{s}\left(x_{(v-1)s+i} - y_i(i)\right)^2. \qquad (6.6)$$

Далі знаходиться середнє значення флуктуацій всіх детрендованих профілів:

$$F_2(s) \equiv \sqrt{\left(\frac{1}{2N_s}\sum_{v=1}^{2N_s} F^2\,(v,s)\right)}. \qquad (6.7)$$

Значення (6.7) можна трактувати як середньоквадратичний зсув (переміщення) точки випадкових блукань у ланцюжку після s кроків.

6.1.4.3 Фрактальна розмірність Хігучі

Фрактальна розмірність Хігучі (Higuchi fractal dimension) [82,83] — це один з різновидів монофрактальної розмірності, яка визначається наступним чином:

Припустимо, що у нас є часовий ряд $x(1), x(2), \ldots, x(N)$ і реконструйований часовий ряд $x_m^k = \{x(m), x(m+k), x(m+2k), \ldots, x(m+[(N-m)/k]\cdot k)\}$ для $m = 1, 2, \ldots, k$, де m представляє початковий час; $k = 2, \ldots, k_{max}$ представляють ступінь часового зміщення. Позначення $[\cdot]$ представляє цілу частину x. Для кожного реконструйованого часового ряду x_m^k розраховується середня довжина часової послідовності $L_m(k)$:

$$L_m(k) = \frac{\sum_{i=1}^{[(N-m)/k]} |x(m+ik) - x(m+(i-1)\cdot k)| \cdot (N-1)}{[(N-m)/k]\cdot k}.$$

Далі, для всіх середніх довжин $L_m(k)$, знаходиться загальне середнє $L(k) = (k)^{-1}\sum_{m=1}^{k} L_m\,(k)$. Згідно методу Хігучі узагальне середнє значення $L(k)$ пропорційне масштабу k, тобто $L(k) \propto k^{-D}$. Далі логарифмуємо обидві сторони й отримуємо рівність $\ln L(k) \propto D \cdot \ln(1/k)$. Інтерполювавши лінію регресії через

залежність $\ln L(k)$ від $\ln(1/k)$, ми можемо отримати показник фрактальності D як кут нахилу цієї лінії. Показник D і представлятиме фрактальну розмірність Хігучі.

6.1.4.4 Фрактальна розмірність Петросяна

Спочатку для часового ряду $\{x_1, x_2, \ldots, y_N\}$ створюємо його дискретизовану (бінарну) версію, z_i:

$$z_i = \begin{cases} 1, & x_i > \langle x \rangle, \\ -1, & x_i \leq \langle x \rangle. \end{cases}$$

Фрактальну розмірність Петросяна (Petrosian fractal dimension) [84–86] можна визначити як

$$D = \log_{10} N / (\log_{10} + \log_{10}[N/(N + 0.4N_\Delta)]),$$

де N_Δ — кількість загальних змін знаку величини z_i: $N_\Delta = \sum_{i=1}^{N-2} |(z_{i+1} - z_i)/2|$.

6.1.4.5 Фрактальна розмірність Каца

Представимо, що сигнал складається з пари точок (x_i, y_i). Тоді, **фрактальна розмірність Каца** (Katz fractal dimension) [87] визначається як

$$D = \log N / (\log N + \log[d/L]),$$

де $L = \sum_{i=0}^{N-2} \sqrt{(y_{i+1} - y_i)^2 + (x_{i+1} - x_i)^2}$, а значення $d = \max\left(\sqrt{(x_i - x_1)^2 - (y_i - y_1)^2}\right)$.

6.1.4.6 Фрактальна розмірність Шевчика

Спочатку, для множини значень (x_i, y_i) виконується нормалізація: $x_i^* = (x_i - x_{min})/(x_{max} - x_{min})$ і $y_i^* = (y_i - y_{min})/(y_{max} - y_{min})$.

Фрактальна розмірність Шевчика (Sevcik fractal dimension) [88] може бути визначена як

$$D = 1 + \ln L / \ln(2 \cdot [N - 1]),$$

L — це довжина сигналу, що може бути розрахована за формулою $L = \sum_{i=0}^{N-2} \sqrt{(y_{i+1}^* - y_i^*)^2 + (x_{i+1}^* - x_i^*)^2}$.

6.1.4.7 Фрактальна розмірність через нормалізовану щільність довжини

Даний показник розраховується в наступний спосіб [89]:

1. Для часового ряду $\{x_1, x_2, \ldots, x_n\}$ виконується стандартизація: $y_i = (x_i - \mu)/\sigma$, де μ — це середнє значення ряду, σ — це стандартне відхилення.
2. Розраховується нормалізована щільність довжини $NLD = N^{-1}\sum_{i=2}^{N}|y_i - y_{i-1}|$. Фактичний розрахунок **фрактальної розмірності через нормалізовану щільність довжини** (fractal dimension via Normalized Length Density, FNLD) базується на побудові монотонної калібрувальної кривої $D = f(NLD)$ за набором функцій Вейєрштрасса, для яких значення D задаються теоретично.
3. Для обчислювальних цілей створено дві моделі цієї залежності:
   – логарифмічну модель: $D = a \cdot \log(NLD - NLD_0) + C$;
   – степеневу модель: $D = a \cdot (NLD - NLD_0)^k$. Бібліотека `neurokit2` використовує саме степеневу модель. Параметр $a = 1.9079$, $k = 0.18383$ і $NLD_0 = 0.097178$, згідно [90].

6.1.4.8 Фрактальна розмірність і спектральна щільність потужності

Фрактальну розмірність можна обчислити на основі аналізу нахилу **спектральної щільності потужності** (power spectral density slope, PSD) [91,92] в сигналах, що характеризуються степеневою частотною залежністю.

Спочатку виконується перетворення часового ряду до частотної області і далі сигнал розбивається на гармонійні коливання 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 через лінійну залежність. 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 $1/f$. Спектральні нахили крутіші за -2 вказують на дробовий броунівський рух, що є проявом процесів випадкового блукання.

6.1.4.9 Кореляційна розмірність

Кореляційна розмірність (correlation dimension, D_2) — це похідна величина від кореляційного інтеграла (кореляційної суми) і може бути подана у вигляді [16–18]:

$$C(\varepsilon) = \frac{1}{N^2} \sum_{\substack{i,j=1 \\ i \neq j}}^{N} \Theta\left(\varepsilon - \| \vec{x}(i) - \vec{x}(j) \|\right), \ \vec{x}(i) \in \Re^m.$$

Кореляційна розмірність може бути виведена з наступної степеневої залежності:

$$C(\varepsilon) \sim \varepsilon^{\nu},$$

або

$$D_2 = \lim_{M \to \infty} \lim_{\varepsilon \to 0} \log(g_\varepsilon / N^2) / \log \varepsilon,$$

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

У першому випадку ми відбираємо i-ту траєкторію та всі інші j-ті траєкторії, і дивимося, чи потрапляють j-ті траєкторії в ε-окіл i-ої траєкторії. Якщо відстань між ними не перевищує окіл радіусом ε, ми ставимо 1. Але якщо відстань між траєкторіями більша за ε, 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, будуть знаходитися досить близько одна від одної. Чим тісніше розташовані точки фазового простору, 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 ε. Ця залежність будується в логарифмічному масштабі.

Ось деякі цікаві приклади.

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 допомагає зрозуміти складність рухів людини під час ходьби чи 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.

6.2 Хід роботи

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

Спочатку імпортуємо необхідні бібліотеки:

```
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

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                   # розмір фонтів рисунку
'lines.linewidth': 2,                # товщина ліній
'axes.titlesize': 'small',           # розмір титулки над рисунком
'axes.labelsize': size,              # розмір підписів по осям
'legend.fontsize': size,             # розмір легенди
'xtick.labelsize': size,             # розмір розмітки по осі Ox
'ytick.labelsize': size,             # розмір розмітки по осі Ox
"font.family": "Serif",              # сімейство стилів підписів
"font.serif": ["Times New Roman"],   # стиль підпису
'savefig.dpi': 300,                  # якість збережених зображень
'axes.grid': False                   # побудова сітки на самому рисунку
}

plt.rcParams.update(params)          # оновлення стилю згідно налаштувань
```

На цьому етапі скористаємось монофрактальними показниками для ідентифікації кризових явищ на ринку золота. Розглянемо ціну золота, наприклад, за період з 1 грудня 2001 року по 1 листопада 2023 року:

```
symbol = 'GC=F'                      # Символ індексу
start = "2001-01-01"                 # Дата початку зчитування даних
end = "2023-11-01"                   # Дата закінчення зчитування даних

data = yf.download(symbol)           # вивантажуємо дані
```

```
time_ser = data['Adj Close'].copy()   # зберігаємо саме ціни закриття

xlabel = 'time, days'                 # підпис по вісі Ox
ylabel = symbol                       # підпис по вісі Oy
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'          # підпис по вісі Ox
ylabel = symbol                    # підпис по вісі Oy
```

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 Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

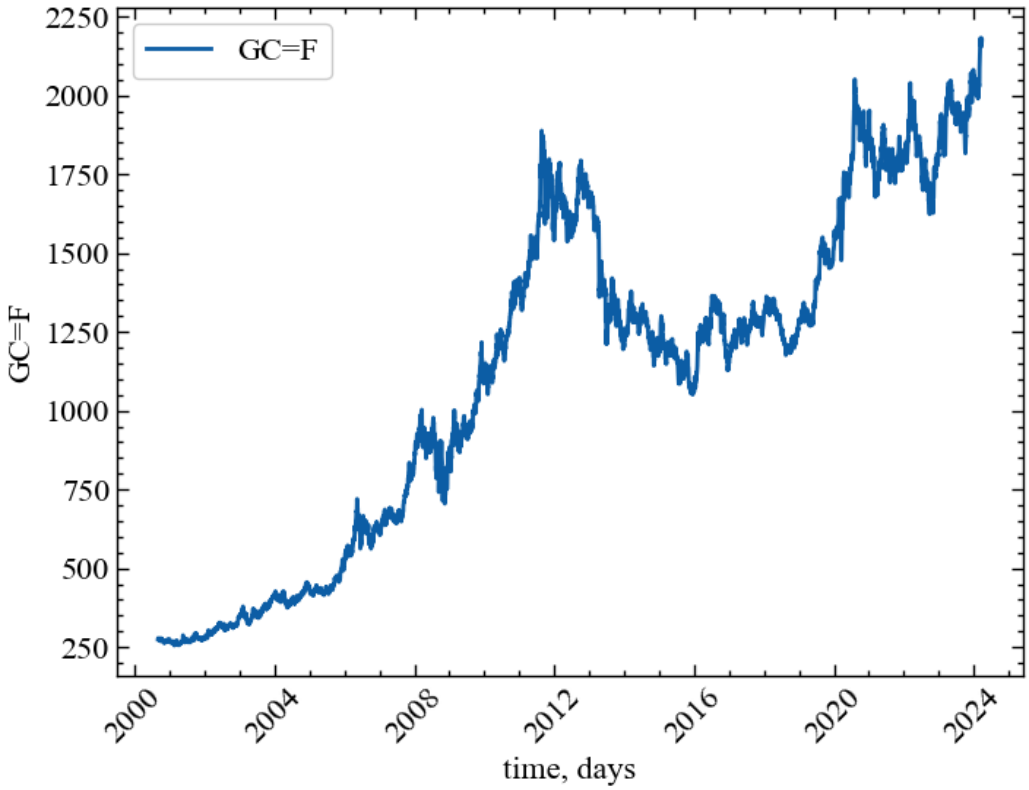


Рис. 6.2: Динаміка щоденних змін ціни золота

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
```

6.2.1 Обчислення показника Херста з використанням *R/S*-аналізу

Для подальших розрахунків використовуватимемо бібліотеку `neurokit2` та `fathon`. Другу з них 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:

`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 [93] відповідно до очікуваного значення для окремих значень (*R/S*);
- **show** (*bool*) — якщо значення `True`, виводить залежність $(R/S)_n$ від n (**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 ціни золота, використовуючи весь часовий ряд. Далі знайдемо значення показника Херста в рамках віконної процедури.

6.2.1.1 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)
```

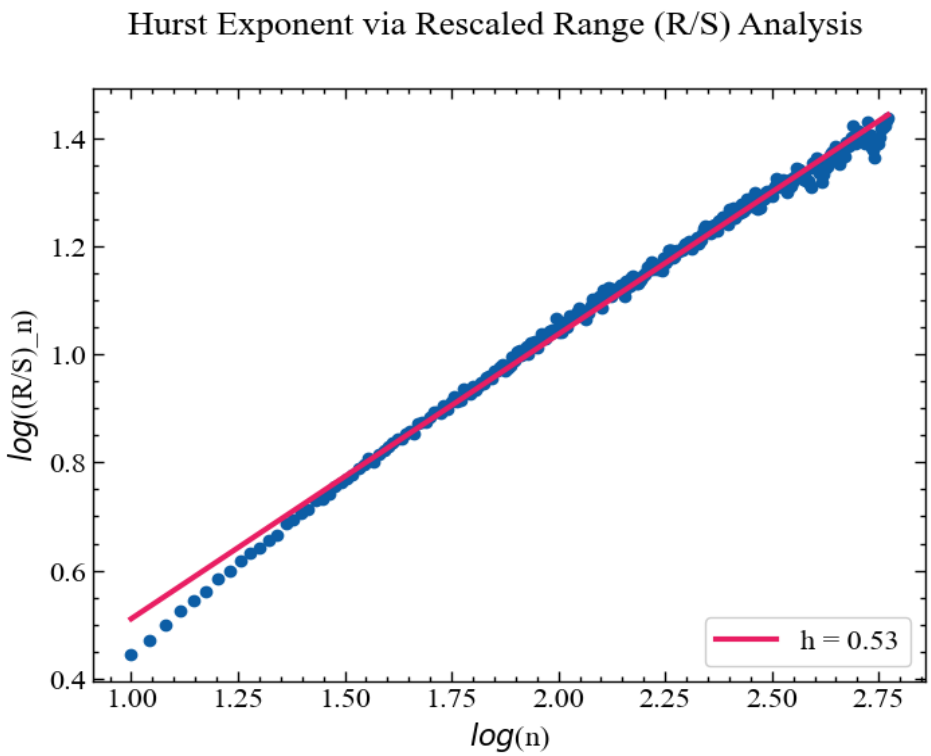


Рис. 6.3: Залежність значень R/S від скейлінгу

Як ми можемо бачити з Рис. 6.3, значення $h = 0.53$, що свідчить про подібність цінової динаміки золота до випадкового блукання. Але оскільки закони, що регулюють ринок, змінюються з часом, мають змінюватись і кореляції всередині системи, а тому коефіцієнт Херста також може змінюватись.

6.2.1.2 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', rotation=45, **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)

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)
```

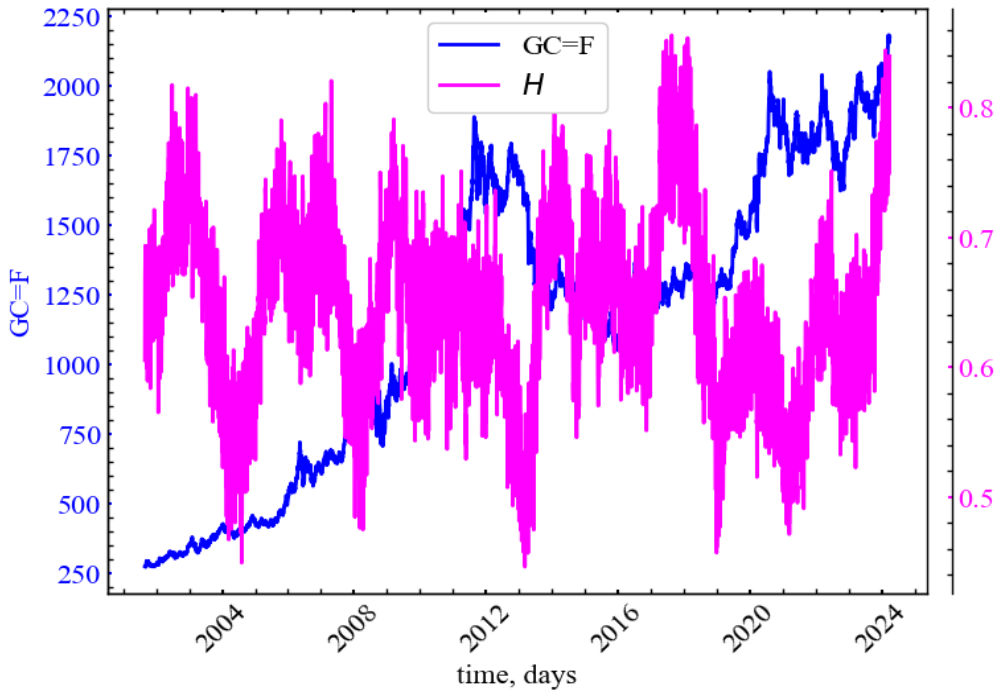


Рис. 6.4: Динаміка ціни золота та показника Херста

На Рис. 6.4 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 трейдерів.

6.2.2 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 роботі.

6.2.2.1 Для всього ряду

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

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 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()
```

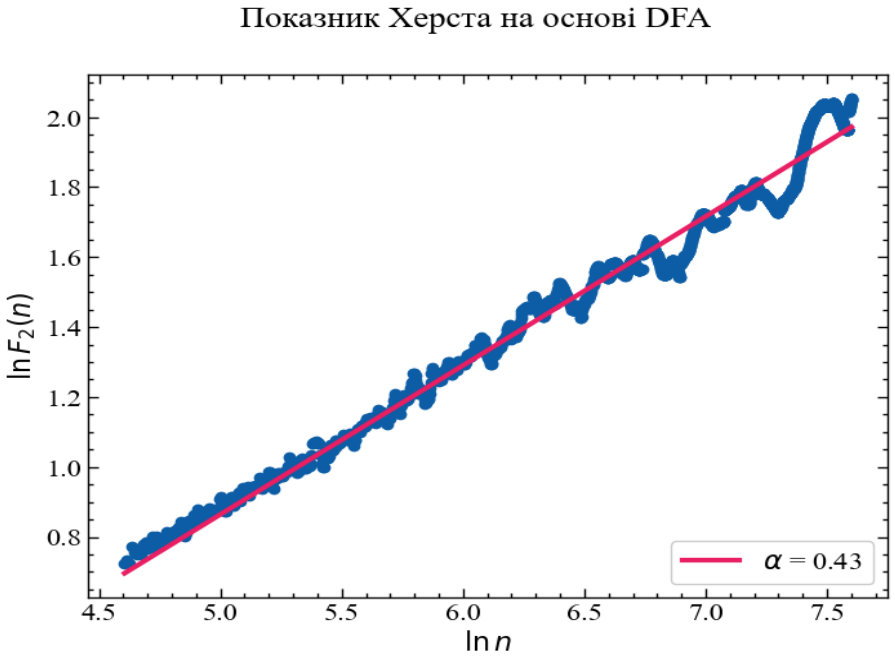


Рис. 6.5: Логарифмічна залежність значень функції флуктуацій від скейлінгу

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

6.2.2.2 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 (α), фрактальну розмірність (D_f), показник спектральної щільності (β) та показник автокореляції (γ):

```
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)
```

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)
```

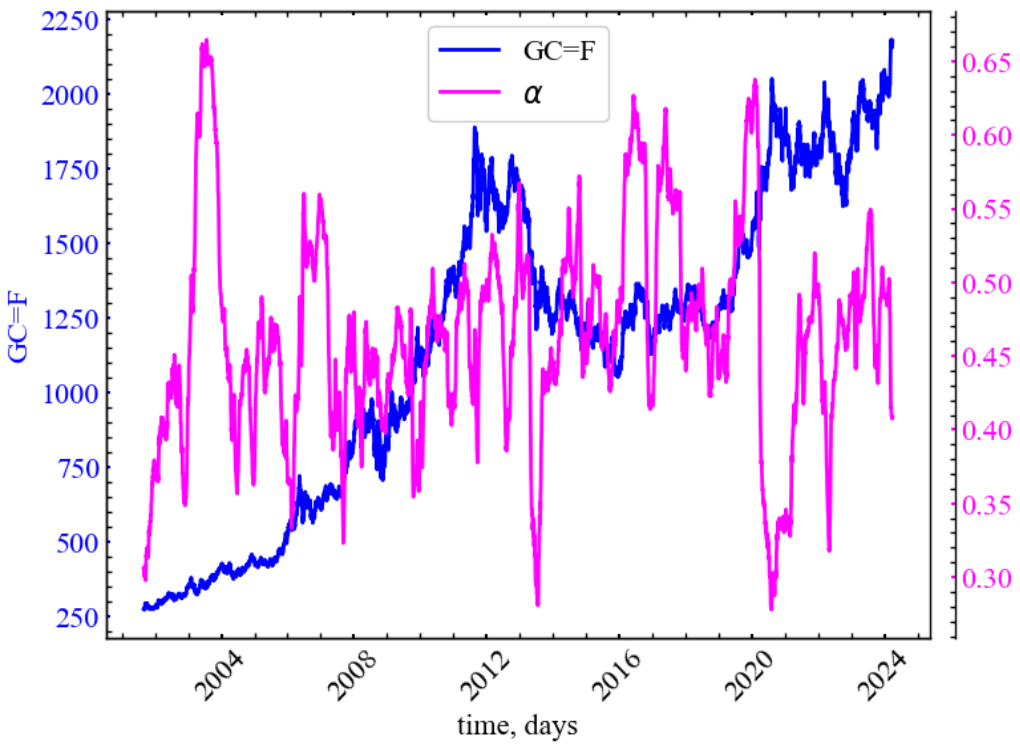


Рис. 6.6: Флуктуації ціни золота та показника альфа

Якщо порівнювати з *R/S*-аналізом, Рис. 6.6 демонструє, що DFA динаміка *узагальненого показника Херста* 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, 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)
```

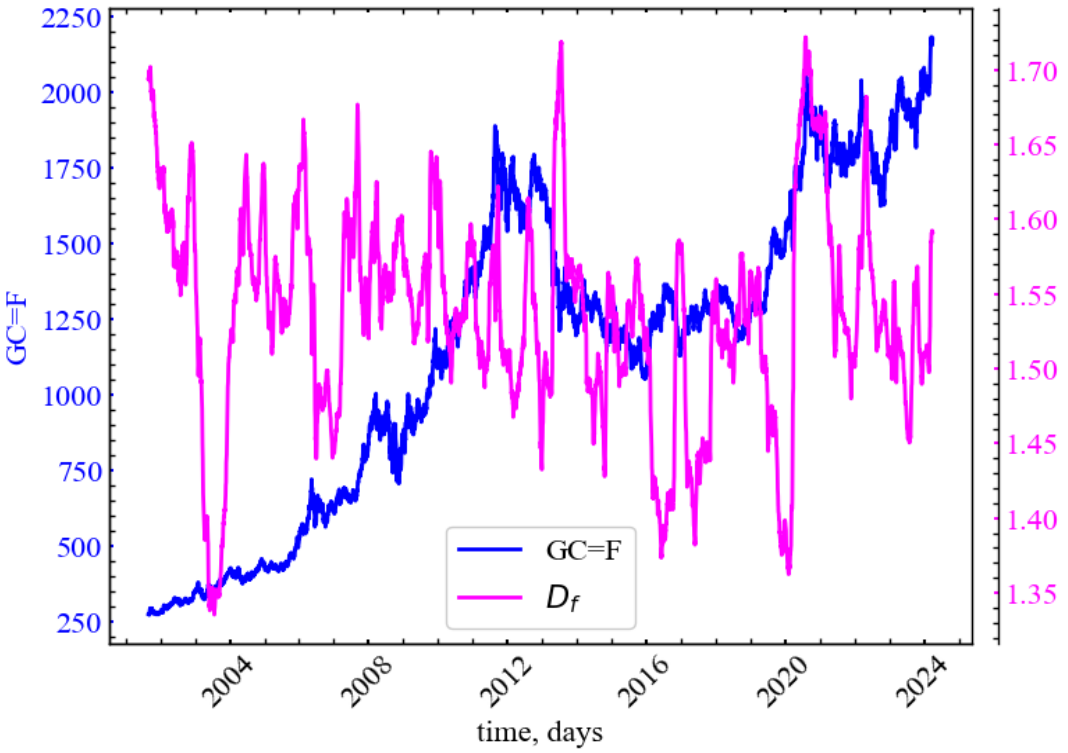


Рис. 6.7: Коливання ціни золота та фрактальної розмірності

Рис. 6.7 показує, що D_f характеризується спадом при кризових станах. Це є індикатором того, що вищий ступень організованості ринку відзеркалюється в більш згладженій або менш волатильних 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)
```

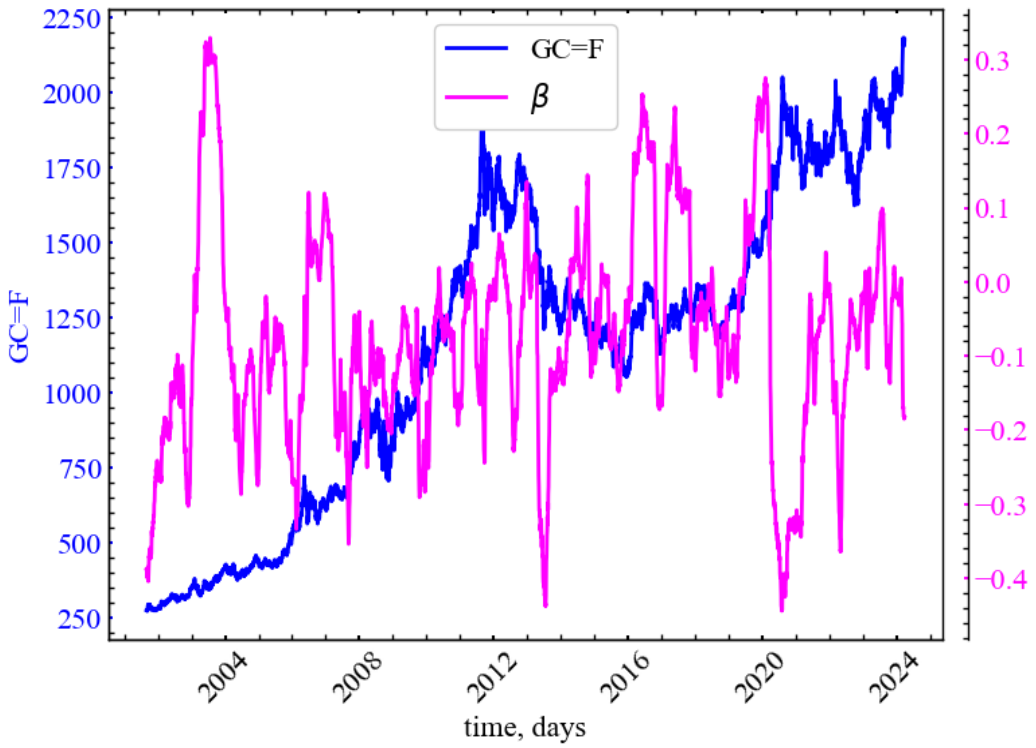


Рис. 6.8: Динаміка ціни золота та показника спектральної щільності

Спектральна густина потужності β (6.5) зростає в кризові періоди, що свідчить про спад потужності сигналу на одиничному інтервалі 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)
```

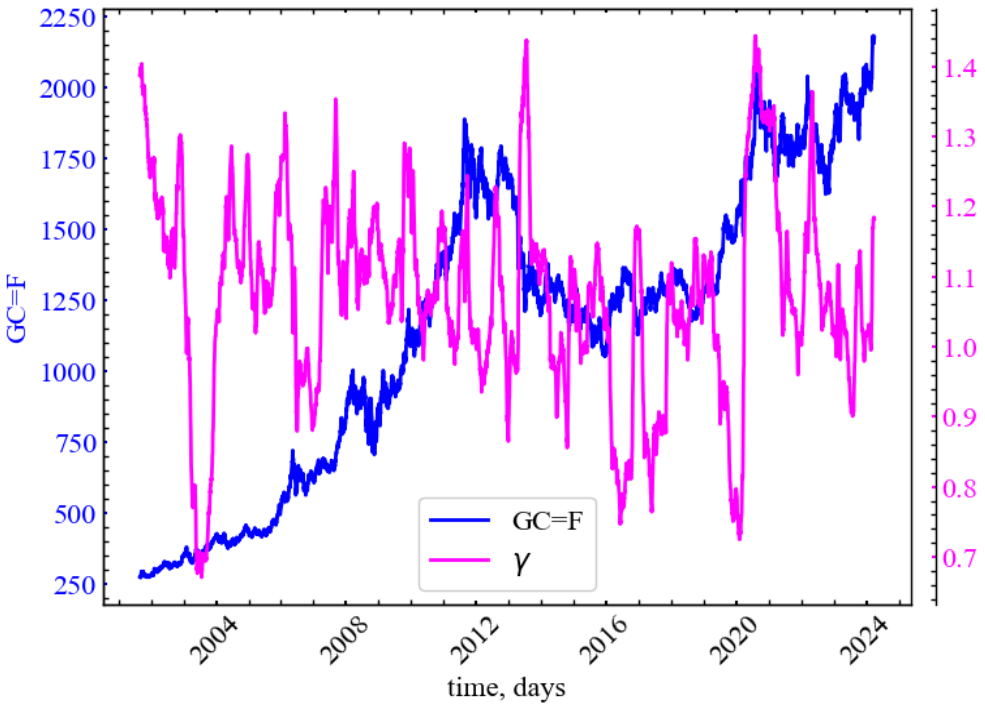


Рис. 6.9: Динаміка індексу золота та показника автокореляції

З Рис. 6.9 видно, що показник γ спадає в кризові та передкризові періоди. Це є показником сповільнення спаду функції автокореляції, що в свою чергу також вказує на зростання корельованності динаміки системи.

6.2.3 Обчислення фрактальної розмірності Хігучі

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

Розрахуємо оптимальне значення k для всього часового ряду. Бібліотека `neurokit2` представляє готову процедуру для автоматизованого підбору даного параметру. Оптимальний k_{max} розраховується на основі точки, в якій значення фрактальної розмірності досягає плато для діапазону значень k_{max} [83].

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;

- k_{max} (*Union[int, str, list], optional*) — максимальна кількість інтервалів (має бути більше або дорівнювати 3), які потрібно перевірити. Якщо k_{max} = `default`, тоді вибирається максимально можливе значення, що відповідає половині довжини сигналу;
- **show** (*bool*) — візуалізовується нахил кривої для обраного значення k_{max}.

Повертає:

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

6.2.3.1 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 k_{max} згідно зазначеної процедури:

```
k_max, info = nk.complexity_k(for_higuchi, k_max=100, show=True)
```

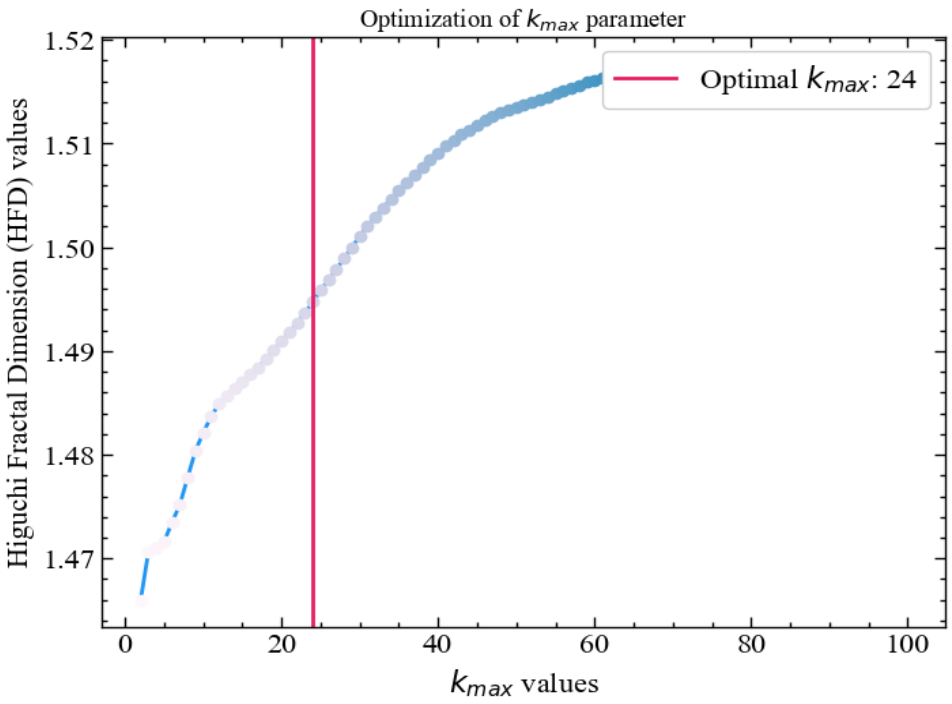


Рис. 6.10: Залежність розмірності Хігучі від діапазону значень kmax

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;
- k_{max} (*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 k_{max}.

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)
```

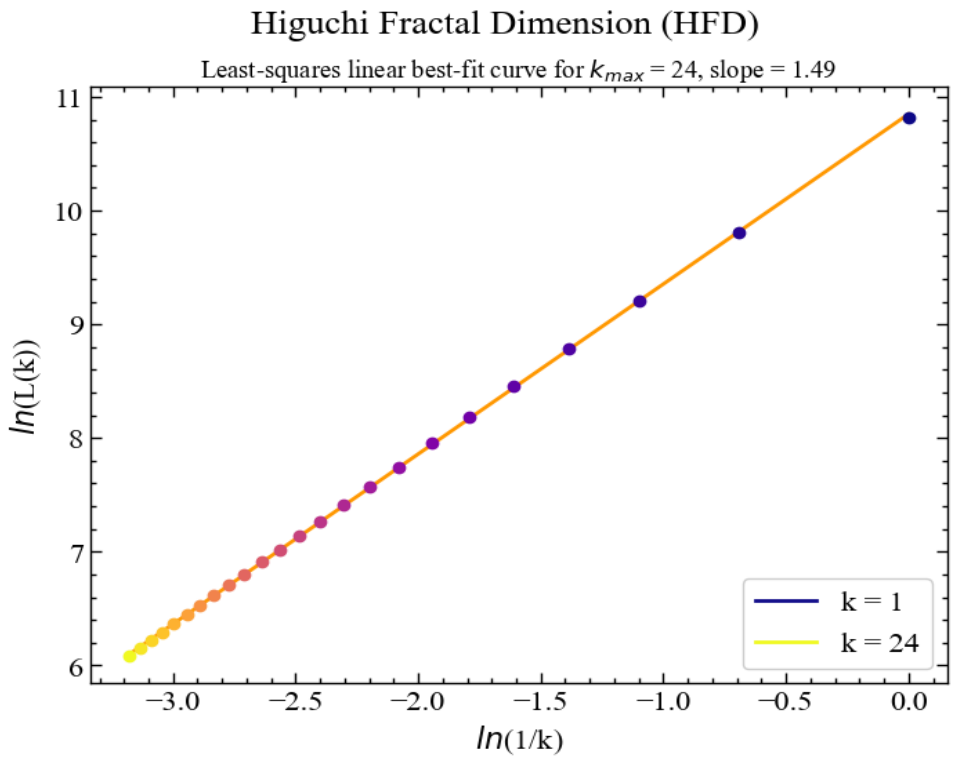


Рис. 6.11: Залежність довжини сигналу від часового зміщення

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

6.2.3.2 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)
```

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)
```

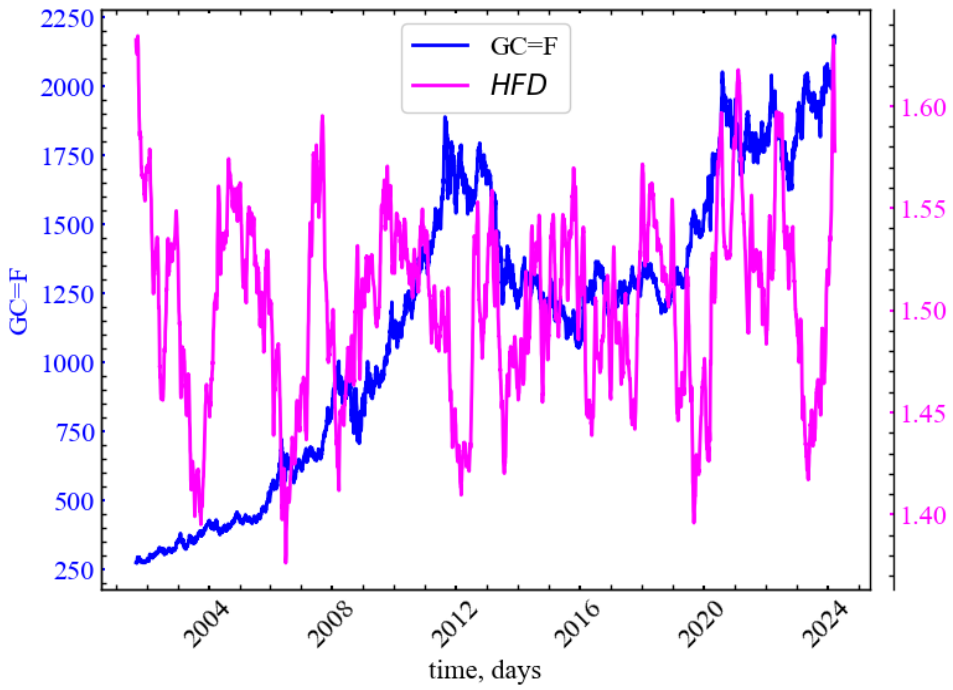


Рис. 6.12: Динаміка індексу золота та фрактальної розмірності Хігучі

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.

6.2.4 Обчислення фрактальної розмірності Петросяна

Петросян [84] запропонував швидкий метод оцінки фрактальної розмірності шляхом перетворення сигналу в двійкову послідовність, з якої оцінюється фрактальна розмірність. Існує кілька варіацій алгоритму (`neurokit2`, наприклад, пропонує варіанти `"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 дискретність вхідного сигналу;
- **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 $\pm 1\sigma$, 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 1σ сигналу, від інших менших змін.

Тепер розглянемо віконну динаміку показника.

6.2.4.1 Віконна процедура

Оскільки більшість з даних методів дискретизації вимагають детрендування ряду, будемо виконувати розрахунки для прибутковостей ціни 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)
```

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)
```

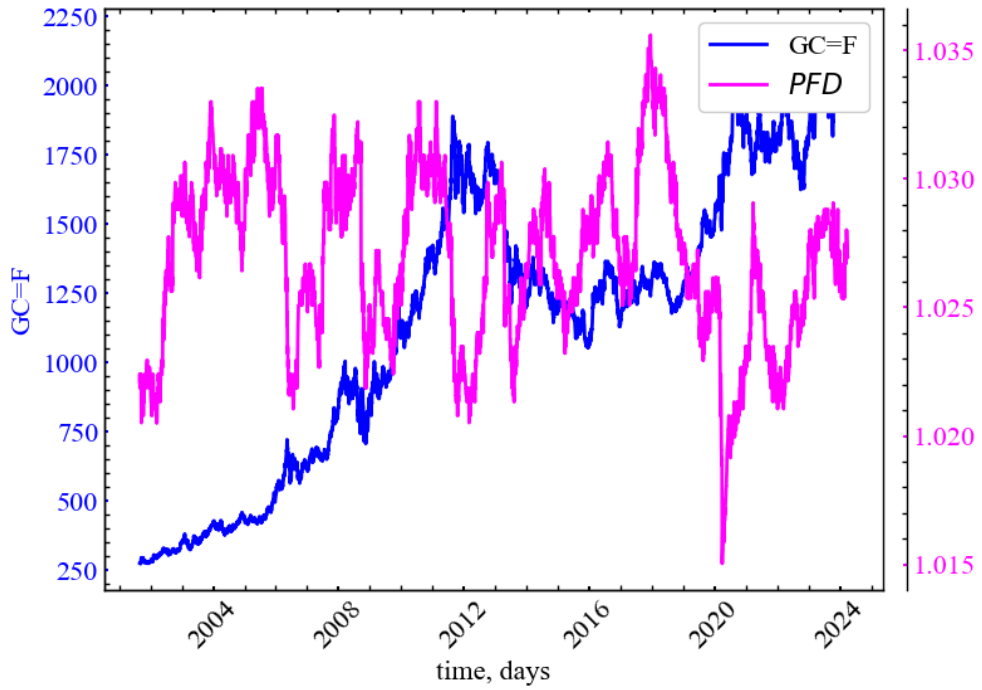


Рис. 6.13: Динаміка ціни золота та фрактальної розмірності Петросяна

Рис. 6.13 показує, що показник Петросяна також спадає під час кризових подій і вказує на зростання періодизації ринку та синхронізації активності трейдерів у відповідні моменти часу.

6.2.5 Обчислення фрактальної розмірності Каца

Обчислимо 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).

6.2.5.1 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)
```

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)
```

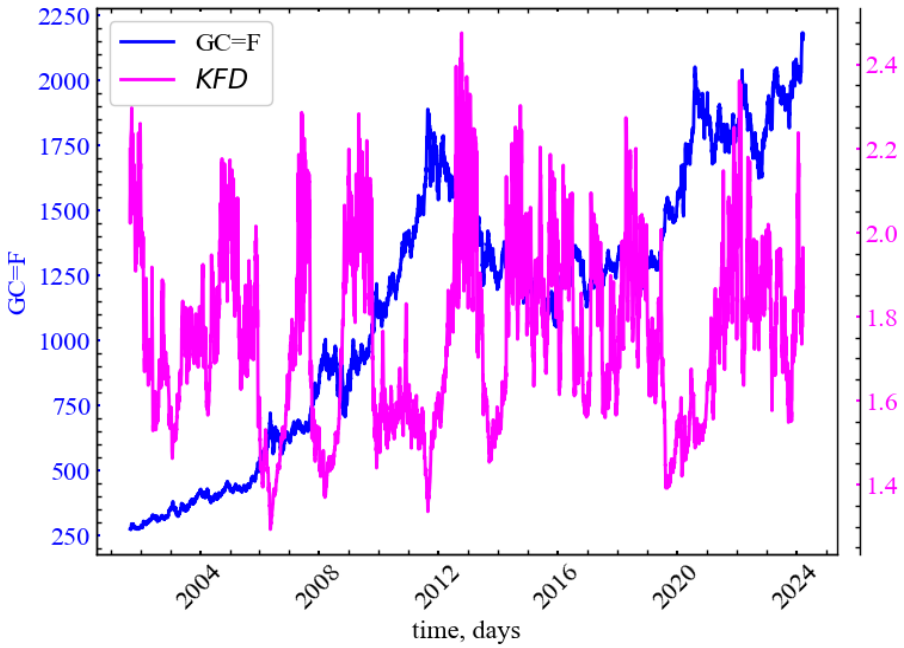


Рис. 6.14: Динаміка ціни золота та фрактальної розмірності Каца

З рисунку видно, що фрактальна розмірність Каца також спадає у кризові та передкризові періоди і також є індикатором зростання ступеня корельованості системи в дані періоди.

6.2.6 Обчислення фрактальної розмірності Шевчика

Алгоритм цієї фрактальної розмірності був запропонований для обчислення фрактальної розмірності сигналів Шевчиком [88]. Цей метод можна використовувати для швидкого вимірювання складності сигналу.

Синтаксис методу:

```
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).

6.2.6.1 Віконна процедура

```
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)
```

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)
```

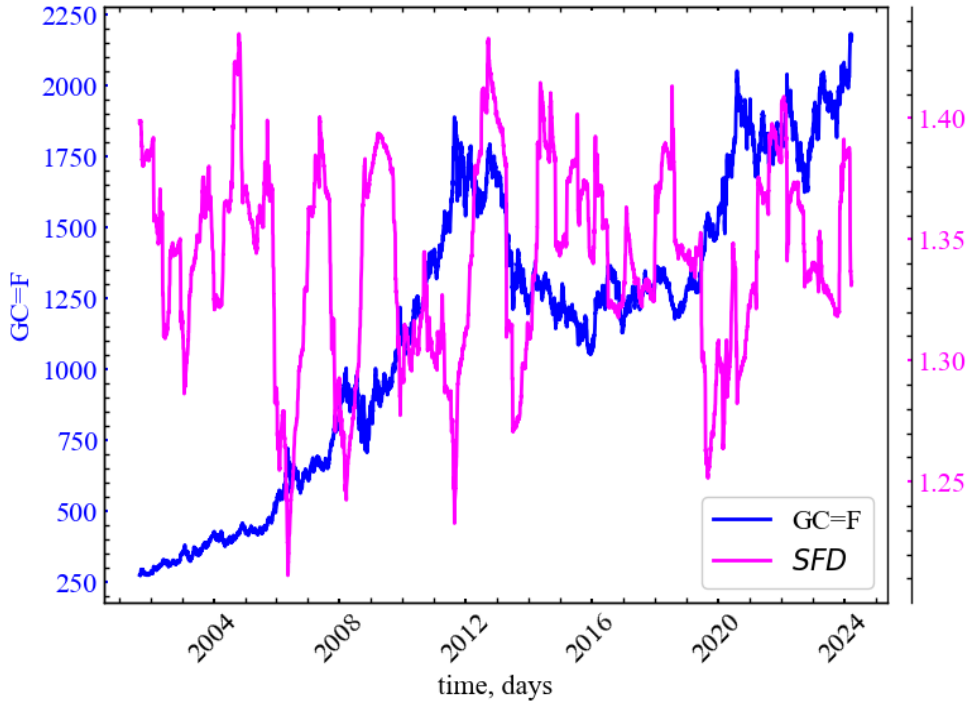


Рис. 6.15: Динаміка ціни золота та фрактальної розмірності Шевчика

Бачимо, що фрактальна розмірність Шевчика реагує спадом на крахові події на ринку золота. Особливо характерними є спади під час криз 2008, 2011, 2015 та 2020 років. Цінові флуктуації золота під час зазначених кризових подій також характеризувалися зростанням персистентності (кореляцій).

6.2.7 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)))`). Метод було розроблено для вимірювання складності сигналів дуже короткої тривалості (< 30 відліків), і його можна використовувати, наприклад, коли цікавлять неперервні зміни фрактальної розмірності сигналу при обчисленні в межах ковзних вікон.

Синтаксис процедури:

```
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;
- **info** (*dict*) — словник із додатковою інформацією по розрахункам.

6.2.7.1 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)
```

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)
```

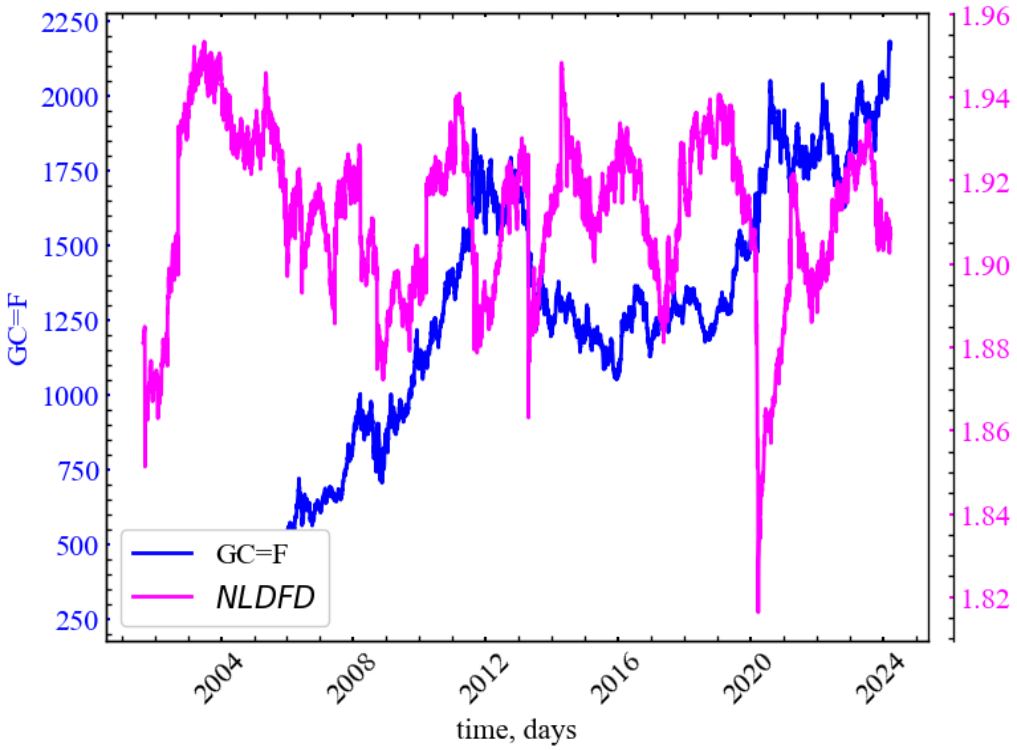


Рис. 6.16: Динаміка ціни золота та фрактальної розмірності через нормалізовану щільність довжини

Рис. 6.16 показує, що *NLDFD* спадає під час кризових та передкризових подій, що свідчить про зростання кореляцій у дані періоди.

6.2.8 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.

6.2.8.1 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)
```

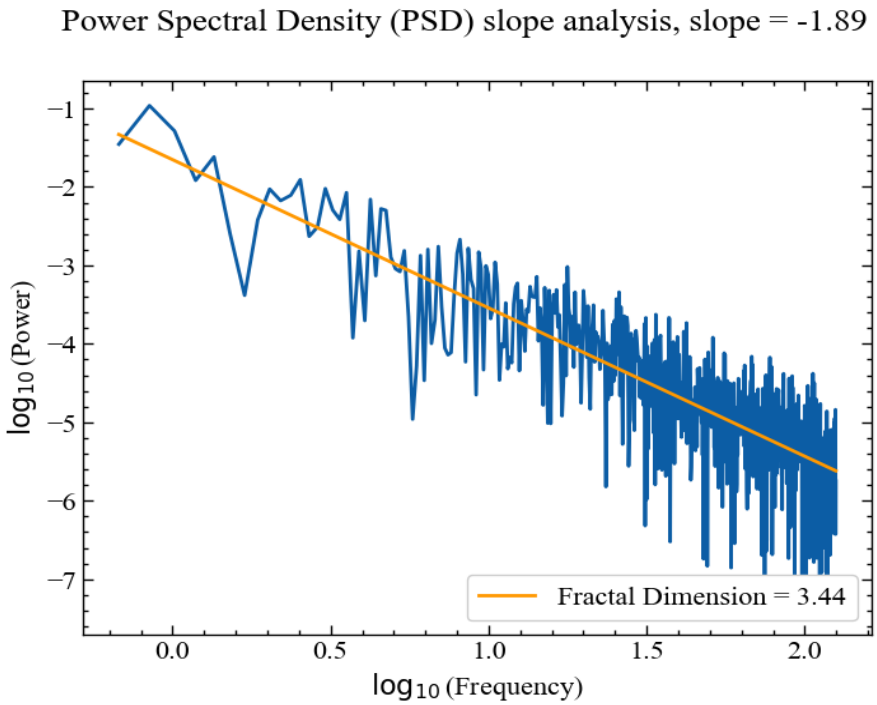


Рис. 6.17: Залежність спектральної щільності потужності від частоти в логарифмічному масштабі

Очевидно, нахил спектральної щільності потужності на різних частотах має лінійну залежність, а кут нахилу прямої побудованої за спектром, 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.

6.2.8.2 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)
```

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)
```

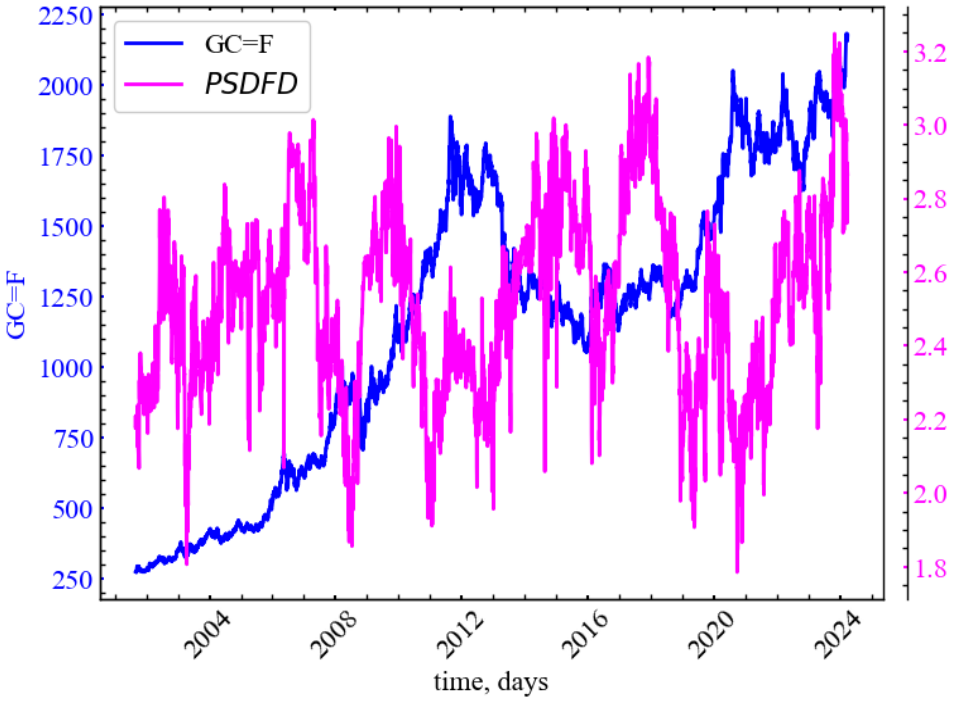


Рис. 6.18: Флуктуації ціни золота та фрактальної розмірності за нахилом спектральної щільності потужності

Рисунок вище показує, що даний показник також реагує спадом на кризові та передкризові події, вказуючи на зростання автокореляції часового ряду. 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.

6.2.9 Обчислення кореляційної розмірності

Кореляційна розмірність є нижньою границею оцінки фрактальної розмірності досліджуваного фазового простору.

Спочатку здійснюється реконструкція фазового простору сигналу 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 у 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 (τ);
- **dimension** (*int*) — розмірність вкладень (m);
- **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.

6.2.9.1 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)
```

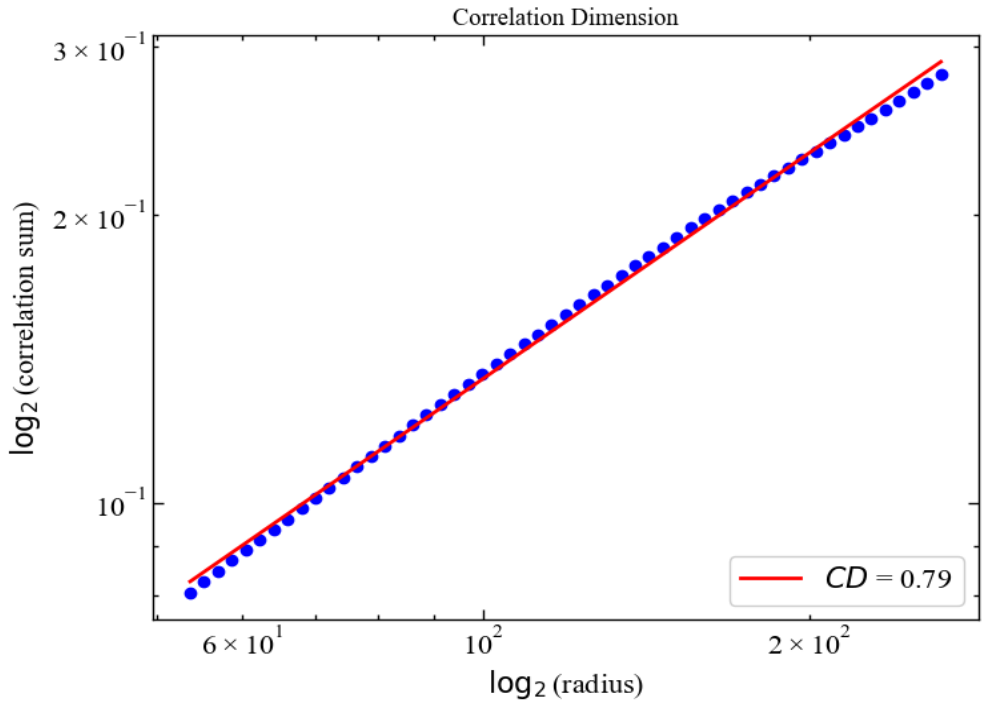


Рис. 6.19: Залежність кореляційної суми від радіусу багатовимірного околу досліджуваних траєкторій

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.

6.2.9.2 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)
```

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)
```

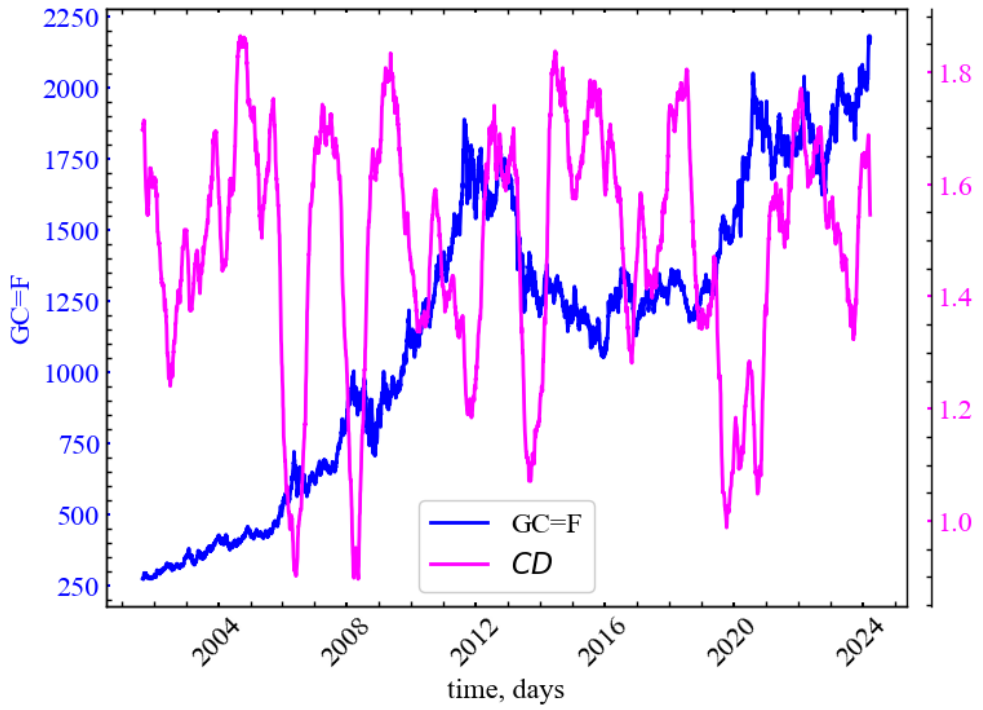


Рис. 6.20: Динаміка ціни золота та кореляційної фрактальної розмірності

Рис. 6.20 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 й 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.

6.3 Висновок

У даній лабораторній роботі було проаналізовано цінові коливання індексу золота з використанням спектра монофрактальних показників. Було показано, що дані методи є досить стійкими до нестаціонарності досілджуваного сигналу. Ринок золота характеризується зростаннями та спадами фрактальної розмірності, що вказує на варіацію ефективності його розвитку в різні моменти часу. Як уже зазначалось, спад фрактальної розмірності в кризовий або передкризовий стан ринку може вказувати на зростання ступеня періодизації (впорядкованості) системи. Зростання фрактальної розмірності може бути

індикатором зростання невпорядкованості. У подальшій лабораторній роботі буде продемонстровано підхід, що дозволяє розраховувати спектр фрактальних показників, які можна отримати на різних просторово-часових масштабах системи.

6.4 Завдання для виконання

1. Вибрати із запропонованої бази даних варіант завдання
2. Виконати дослідження фрактальних характеристик заданих часових рядів. Зберегти результати в окремому файлі
3. Порівняти значення коефіцієнтів Херста, одержаних методом R/S-аналізу та DFA
4. Провести повний аналіз різних методів розрахунку монофрактальних розмірностей
5. Дати інтерпретацію отриманим результатам

7. Лабораторна робота № 7

Тема. Мультифрактальний аналіз складних систем

Мета. Навчитися використовувати мультифрактальний аналіз детрендованих флуктуацій для отримання індикаторів нелінійних змін у системі

7.1 Теоретичні відомості

7.1.1 Визначення 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 D, 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 D, такі фрактали характеризуються й деякими специфічними статистичними властивостями.

Простіше всього пояснити, що розуміється під "неоднорідним фракталом" на прикладі трикутника Серпінського, отриманого за допомогою методу випадкових ітерацій (Рис. 7.1).

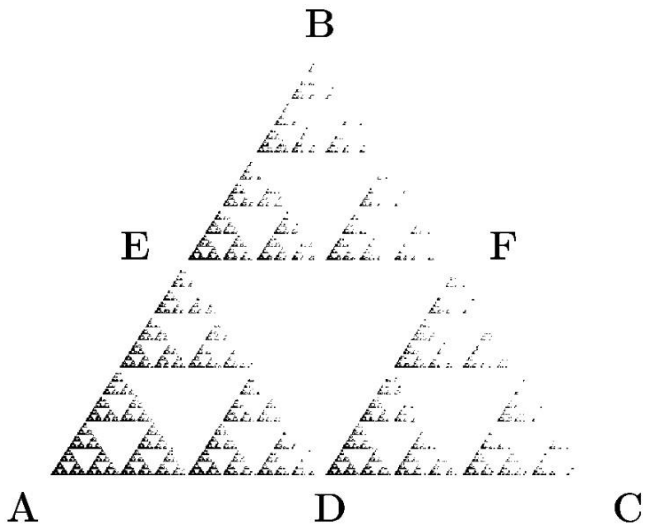


Рис. 7.1: Трикутник Серпинського, області якого згенеровані з різними ймовірностями

Припустимо, що в методі випадкових ітерацій ми тепер із якоїсь причини віддали перевагу одній із вершин трикутника, наприклад, вершині A, і стали

вибирати її з імовірністю 90%. Дві ж інші вершини B і C 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 ABC розподілені тепер вкрай нерівномірно. Більша їх частина перебуває біля вершини A та її прообразів. Водночас у вершин B і C (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 фрактала — самоподібність. Дійсно, трикутник DFC, хоча в ньому у 20 разів менше точок, за своїми статистичними і геометричними властивостями повністю подібний до великого трикутника ABC. Так само, як і у великому трикутнику, точки в ньому концентруються здебільшого поблизу вершини D — аналогу вершини A.

Рис. 7.2 більш детально демонструє результуючий розподіл точок по трикутнику Серпінського. Цифри в кожному з маленьких трикутників показують його відносну заселеність точками множини.

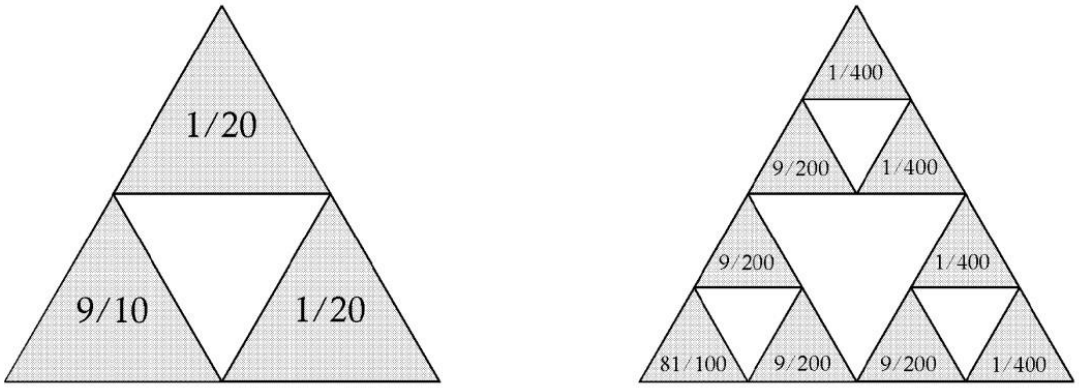


Рис. 7.2: Розподіл точок по трикутнику Серпінського, представленого на попередньому рисунку

Однак, не дивлячись на нерівномірність розподілу точок фрактала, фрактальна розмірність залишилась при цьому такою ж, $D = \ln 3/\ln 2$. 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 M. На наступному

(першому) етапі, у центрі рисунка, показано, як ту саму множину можна покрити трьома меншими квадратами зі сторонами $l_1 = 1/2,\ l_2 = l_3 = 5/16$, у яких, відповідно, міститься частка $p_1 = 1/2, p_2 = 1/3$ та $p_3 = 1/6$ усіх точок.

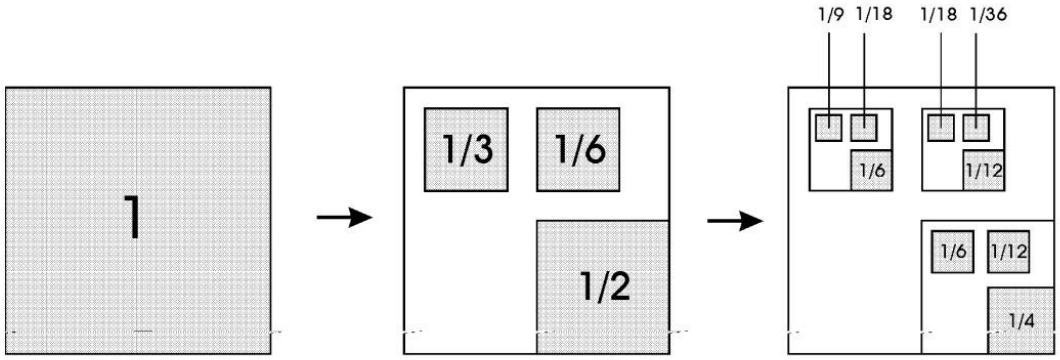


Рис. 7.3: Приклад мультифрактала, що підкоряється ренормалізаційній схемі

Наступний етап покриття (зображений на рисунку праворуч) містить уже 9 квадратів зі сторонами $l_1^2 = 1/4$, $l_1 l_2 = l_1 l_3 = 5/32$ (у нижньому правому куті) і $l_2 l_1 = 5/32$, $l_2^2 = l_2 l_3 = 25/256$ (угорі праворуч і ліворуч). Відносна заселеність цих квадратів точками множини показана на рисунку. Вона відповідає добутку чинників заселеності (ймовірностей): $p_1^2 = 1/4$, $p_1 p_2 = 1/6$, $p_1 p_3 = 1/12$ — для нижньої правої групи, $p_2 p_1 = 1/6$, $p_2^2 = 1/9$, $p_2 p_3 = 1/18$ — для верхньої лівої та $p_3 p_1 = 1/12$, $p_3 p_2 = 1/18$, $p_3^2 = 1/36$ — для верхньої правої групи. Зазначимо, що є чітка відповідність між заселеністю квадрата $p_j p_i$ і його розмірами $l_i l_i$.

Подальший процес розбиття і покриття множини M здійснюється згідно із цією ренормалізаційною схемою. Кожен квадрат, що має на n-му кроці розмір l і заселеність p, замінюється на $n+1$ кроці трьома квадратами з розмірами ll_1, ll_2, ll_3 і заселеностями pp_1, pp_2, pp_3 відповідно, розміщеними таким самим чином відносно один одного, як показано на Рис. 7.3.

Двоє з 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** (multifractals), і їх вивченням ми й займемося надалі.

7.1.2 Узагальнені фрактальні розмірності D_q

Дамо загальне визначення мультифрактала. Розглянемо фрактальний об'єкт, що займає якусь обмежену ділянку Ω розміру L у Евклідовому просторі з розмірністю d. Нехай на якомусь етапі його побудови він являє собою множину з $N \gg 1$ точок, якось розподілених у цій області. Ми будемо припускати, що врешті-решт $N \to \infty$. 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 Ω на кубічні клітинки зі стороною $\varepsilon \ll L$ та об'ємом ε^d. Далі нас будуть цікавити тільки зайняті клітинки, у яких міститься хоча б одна точка. Нехай номер зайнятих комірок i змінюється в межах $i = 1,2,\ldots,N(\varepsilon)$, де $N(\varepsilon)$ — сумарна кількість зайнятих клітинок, яка, звісно, залежить від розміру клітинки ε.

Нехай $n_i(\varepsilon)$ представляє собою кількість точок у клітинці з номером i, тоді величина $p_i(\varepsilon) = \lim_{N\to\infty} n_i(\varepsilon)/N$ представляє собою ймовірність того, що навмання взята точка з нашої множини знаходиться в комірці i. Інакше кажучи, ймовірності p_i характеризують відносну заселеність комірок. З умови нормування ймовірності випливає, що

$$\sum_{i=1}^{N(\varepsilon)} p_i\,(\varepsilon) = 1.$$

Уведемо тепер у розгляд **узагальнену статистичну суму** $Z(q,\varepsilon)$, що характеризується показником ступеня q, який може набувати будь-яких значень в інтервалі $-\infty < q < +\infty$

$$Z(q,\varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q\,(\varepsilon).$$

Спектр **узагальнених фрактальних розмірностей** (generalized fractal dimensions) D_q, що характеризує даний розподіл точок в області Ω, визначається за допомогою співвідношення $D_q = \tau(q)/(q-1)$, де функція $\tau(q)$ має вид

$$\tau(q) = \lim_{\varepsilon\to 0} \ln Z(q,\varepsilon) \,/\, \ln \varepsilon.$$

Як ми покажемо нижче, якщо $D_q = D = \text{const}$, тобто не залежить від q, то дана множина точок являє собою звичайний, регулярний фрактал, який характеризується лише однією величиною — фрактальною розмірністю D. Навпаки, якщо функція D_q якось змінюється з q, то розглянута множина точок представляє мультифрактал.

Таким чином, мультифрактал у загальному випадку характеризується деякою **нелінійною** функцією $\tau(q)$, що визначає поведінку статистичної суми $Z(q,\varepsilon)$ при $\varepsilon \to 0$:

$$Z(q,\varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q\,(\varepsilon) \approx \varepsilon^{\tau(q)}. \qquad (7.1)$$

Слід мати на увазі, що в реальній ситуації ми завжди маємо скінченне, хоча й дуже велике число дискретних точок N, тому при комп'ютерному моделюванні конкретної множини граничний перехід $\varepsilon \to 0$ треба виконувати з обережністю, пам'ятаючи, що йому завжди передує ліміт $N \to 0$.

Покажемо тепер, як поводиться узагальнена статистична сума у випадку звичайного регулярного фрактала з фрактальною розмірністю D. У цьому випадку в усіх зайнятих комірках міститься однакова кількість точок, $n_i(\varepsilon) = N/N(\varepsilon)$, тобто фрактал представляється **однорідним**. Тоді очевидно, що відносні населеності клітинок, $p_i(\varepsilon) = 1/N(\varepsilon)$, також однакові, і узагальнена статистична сума набуває вигляду

$$Z(q,\varepsilon) = N^{1-q}(\varepsilon). \qquad (7.2)$$

Врахуємо тепер, що, згідно визначеню фрактальної розмірності D, кількість зайнятих клітинок при достатньо малому ε поводить себе наступним чином:

$$N(\varepsilon) \approx \varepsilon^{-D}. \qquad (7.3)$$

Підставляючи (7.3) у (7.2), і порівнюючи з (7.1), отримуємо

$$\varepsilon^{\tau(q)} = \varepsilon^{-D(1-q)} \to \tau(q) = (q-1)D. \qquad (7.4)$$

Ми приходимо до висновку, що у випадку звичайного фрактала функція (7.4) є лінійною. Тоді всі D_q дійсно не залежать від q. Фрактал у якого всі узагальнені фрактальні розмірності D_q співпадають називається **монофракталом** (monofractal).

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 D_q, кількість котрих, у загальному випадку, нескінченна.

Так, наприклад, при $q \to +\infty$ основний внесок в узагальнену статистичну суму (7.1) вносять комірки, що містять найбільшу кількість частинок n_i у них і, відповідно, що характеризуються найбільшою ймовірністю їх заповнення p_i. Навпаки, при $q \to -\infty$ основний внесок в узагальнену статистичну суму вносять найбільш розрідженні комірки з найменшою ймовірністю їх заповнення p_i. Таким чином, функція D_q показує, наскільки неоднорідним представляється досліджувана множина точок Ω.

У подальшому для характеристики розподілу точок необхідно знати не тільки функцію $\tau(q)$, але і її похідну:

$$\frac{d\tau(q)}{dq} = \lim_{\varepsilon \to 0} \sum_{i=1}^{N(\varepsilon)} p_i^q \ln p_i \Big/ \left(\sum_{i=1}^{N(\varepsilon)} p_i^q \right) \ln \varepsilon.$$

Ця похідна має важливий фізичний зміст, який буде продемонстровано пізніше. Зараз знову зазначимо, що для мультифрактальної системи вона не залишається константною і змінюється з q.

7.1.3 Функція мультифрактального спектра $f(\alpha)$

7.1.3.1 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 D_q, де q приймає будь-які значення в інтервалі $-\infty < q < +\infty$. Однак величини D_q 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** (multifractal spectrum) $f(\alpha)$ (спектр сингулярностей мультифрактала), до якої, як ми побачимо надалі, більше підходить термін фрактальна розмірність. Ми покажемо, що величина $f(\alpha)$ фактично дорівнює хаусдорфовій розмірності якоїсь однорідної фрактальної підмножини із вихідної множини Ω, що дає домінантний внесок у статистичну суму при заданій величині q.

Однією з основних характеристик мультифрактала є набір імовірностей p_i, що показують відносну заселеність клітинок ε, якими ми покриваємо цю множину. Чим менший розмір клітинки, тим менша величина її заселеності. Для самоподібних множин залежність p_i від розміру клітинки ε має степеневий характер:

$$p_i(\varepsilon) \approx \varepsilon^{\alpha_i},$$

де α_i являє собою деякий показник ступеня (різний для різнок клітинок i).

> Додатково по α
>
> Спрямовуючи значення ε до нуля, фрактальність можна розглядати локально для кожної точки (елемента) досліджуваної системи, і таким чином показник α є локальною фрактальною розмірністю. Його ще називають *показником Гьолдера* або *силою сингулярності*.
>
> Можемо спостерігати саме степеневу залежність, оскільки, вочевидь, розподіл маси (флуктуацій) концентрується з різною "силою" α, тож і ймовірнісна міра змінюється пропорційно розмірам вікон ε

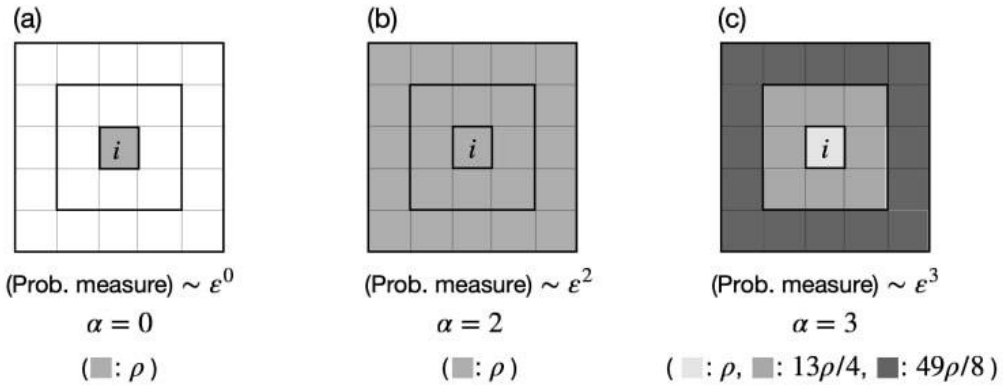


Рис. 7.4: Схематичне представлення залежності сили сингулярності та густини порівняно з околицями

Сірий масштаб являє собою ймовірнісну міру для кожної локації, як показано на кожній панелі. На рисунку (a) тільки i-та локація має ненульову щільність, інші місця порожні. Імовірнісна міра на комірці залишається ρ, навіть коли розмір клітинки ε збільшується. ε збільшується, що підкреслюється жирною лінією. Проте, через те, що далі ми не спостерігаємо зростання щільності, показник α залишається нульовим. На Рис. 7.22 (b) усі комірки мають однакову щільність. Імовірнісні міри комірок дорівнюють ρ, 9ρ і 25ρ для

найменшої, другої найменшої та найбільшої комірки (виділено жирною лінією). Таким чином, сила сингулярності i-го осередку дорівнює 2. На Рис. 7.22 (с) i-й осередок є розрідженим порівняно з навколишніми осередками. Імовірнісна міра осередків дорівнює ρ, 27ρ і 125ρ для найменшого, другого найменшого і найбільшого осередку (виділено жирною лінією). Таким чином, сила сингулярності i-ої комірки дорівнює 3.

> Додатково по α
>
> Можна сказати, що чим більш гладкою видається поверхня системи, чим менше елементів задіяно в її розвитку, тим меншим є показник сингулярності. Що більше елементів системи вступають у взаємозв'язок один з одним, що більше процесів протікає під час еволюції системи, то більшим є показник сингулярності

Відомо, що для регулярного (однорідного) фрактала всі показники ступеня α_i однакові й рівні фрактальній розмірності D:

$$p_i = 1/N(\varepsilon) \approx \varepsilon^D.$$

У даному випадку статистична сума (7.1) приймає наступний вигляд:

$$Z(q,\varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q\,(\varepsilon) = N(\varepsilon)\varepsilon^{Dq} = \varepsilon^{-D}\varepsilon^{Dq} \approx \varepsilon^{D(q-1)}.$$

Тому $\tau(q) = D(q-1)$ і всі узагальнені фрактальні розмірності $D_q = D$ у цьому випадку співпадають та не залежать від q.

Однак, для такого складного об'єкта, як мультифрактал, унаслідок його неоднорідності, ймовірності заповнення клітинок p_i у загальному випадку різняться, і показник ступеня α_i для різних клітинок може приймати різні значення. Достатньо типовою є ситуація, коли ці значення неперервно заповнюють деякий закритий інтервал $(\alpha_{min}, \alpha_{max})$, причому $p_{min} \approx \varepsilon^{\alpha_{max}}$, а $p_{max} \approx \varepsilon^{\alpha_{min}}$.

Тепер перейдемо до питання розподілу ймовірностей різних значень α_i. Нехай $n(\alpha)d\alpha$ є ймовірністю того, що α_i знаходиться в інтервалі від α до $\alpha + d\alpha$. Іншими словами, $n(\alpha)d\alpha$ представляє собою відносну кількість клітинок i, що характеризуються однією і тією самою мірою p_i з α_i, що лежать у цьому інтервалі. У випадку монофрактала, для котрого всі α_i однакові (і рівні фрактальній розмірності D), це число, очевидно, пропорційно повній кількості

клітинок $N(\varepsilon) \approx \varepsilon^{-D}$, степеневим чином залежних від розміру клітинки ε. Показник степеня в цьому співвідношені визначається фрактальною розмірністю множини D.

Для мультифрактала, однак, це не так, і різні значення α_i зустрічаються з імовірністю, що характеризується не однією і тією ж величиною D, а різними (в залежності від α) значеннями показника $f(\alpha)$:

$$n(\alpha) \approx \varepsilon^{-f(\alpha)}. \tag{7.5}$$

Таким чином, фізичний сенс функції $f(\alpha)$ полягає в тому, що вона представляє собою розмірність Хаусдорфа деякої однорідної підмножини Ω_α із вихідної множини Ω, що характеризується однаковими ймовірностями заповнення клітинок $p_i \approx \varepsilon^\alpha$. Оскільки фрактальна розмірність підмножини очевидно завжди менша або рівна фрактальній розмірності вихідної множини D_0, має місце важлива нерівність для функції $f(\alpha)$:

$$f(\alpha) \leq D_0.$$

У результаті можна зробити висновок, що множина різних значень функції $f(\alpha)$ (при різних α) представляє собою **спектр фрактальних розмірностей** [94,95] однорідних підмножин Ω_α, на які можна розбити вихідну множину Ω, кожна з яких характеризується власним значенням фрактальної розмірності $f(\alpha)$.

7.1.3.2 Перетворення Лежандра

Встановимо зв'язок функції $f(\alpha)$ із введенною раніше функцією $\tau(q)$. Обчислимо для цього статистичну суму $Z(q,\varepsilon)$. Підставляючи у $Z(q,\varepsilon)$ ймовірності $p_i \approx \varepsilon^{\alpha_i}$, та переходячи від підсумовування по i до інтегрування по α з щільністю ймовірностей (7.5), отримаємо

$$Z(q,\varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q\,(\varepsilon) \approx \int d\alpha n(\alpha)\varepsilon^{q\alpha} \approx \int d\alpha \varepsilon^{q\alpha - f(\alpha)}. \tag{7.6}$$

Так як величина ε дуже мала, основний внесок у цей інтеграл вноситимуть ті значення $\alpha(q)$, при яких показник степеня $q\alpha - f(\alpha)$ виявляється мінімальним (а підінтегральна функція — максимальною). Цей вклад буде пропорційним значенню підінтегральної функції у точці максимума. Саме ж значення $\alpha(q)$ визначається при цьому з наступної умови:

$$\frac{d}{d\alpha}[q\alpha - f(\alpha)]\Big|_{\alpha=\alpha(q)} = 0.$$

Також, з умови мінімуму ми маємо

$$\frac{d^2}{d\alpha^2}[q\alpha - f(\alpha)]\Big|_{\alpha=\alpha(q)} > 0.$$

У результаті отримуємо, що залежність $\alpha(q)$ неявним чином визначається з $q = df(\alpha)/d\alpha$, і що функція $f(\alpha)$ усюди є випуклою:

$$f''(\alpha) > 0.$$

Це значить, що величина $f\big(\alpha(q)\big)$ дійсно є фрактальною розмірністю підмножини $\Omega_{\alpha(q)}$, що має найбільший домінуючий внесок у статистичну суму (7.6) при заданій величині показника ступеня q.

Оскільки $Z(q,\varepsilon) = \tau(q)$, приходимо до висновку, що

$$\tau(q) = q\alpha(q) - f\big(\alpha(q)\big). \tag{7.7}$$

Пам'ятаючи, що $\tau(q) = D(q-1)$, можемо віднайти функцію D_q:

$$D_q = \frac{1}{q-1}\big[q\alpha(q) - f\big(\alpha(q)\big)\big]. \tag{7.8}$$

Таким чином, якщо ми знаємо функцію мультифрактального спектра $f(\alpha)$, то за домомогою співвідношень (7.8) та (7.9) ми можемо знайти функцію D_q. Навпаки, знаючи D_q, ми можемо відтворити залежність $\alpha(q)$ за допомогою рівняння

$$\alpha(q) = \frac{d}{dq}\big[(q-1)D_q\big] \tag{7.9}$$

і після цього знайти із (7.9) $f\big(\alpha(q)\big)$. Ці два рівняння і визначають функцію $f(\alpha)$.

$$\frac{d\tau}{dq}\frac{dq}{d\alpha} = q + \alpha\frac{dq}{d\alpha} - \frac{df}{d\alpha}.$$

Приймаючи до уваги, що $q = df/d\alpha$, і скорочуючи це рівняння на $dq/d\alpha$, приходимо до співвідношення $\alpha = d\tau(q)/dq$, яке еквівалентне (7.9).

Вирази для $\tau(q)$ та $\alpha(q)$ задають **перетворення Лежандра** (Legendre transformation) [18,96] від змінних $\{q, \tau(q)\}$ до змінних $\{\alpha, f(\alpha)\}$: $\alpha = d\tau/dq$ та $f(\alpha) = q(d\tau/dq) - \tau(q)$. Як відомо, для однорідного фрактала $D_q = D = \text{const}$. Тому $\alpha = d\tau/dq = D$ і $f(\alpha) = q\alpha - \tau(q) = qD - D(q-1) = D$. У цьому випадку "графік" функції $f(\alpha)$ на площині $\big(\alpha, f(\alpha)\big)$ складається лише з однієї точки (D, D).

7.1.4 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. 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) [97–99], котрий і досі залишається одним із найпотужніших методів для аналізу систем різної природи та складності [100–113].

Побудова MFDFA складається з 9 кроків:

1. *"Шум і випадкові блукання у часовому ряді"* представляє метод приведення часового ряду до такого, що подібний до випадкового блукання.
2. *"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.

3. *"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.
4. *"Локальне детрендування часового ряду"* представляє обчислення такого ж локального середньоквадратичного відхилення навколо трендів, які часто зустрічаються у фінансових часових рядах.
5. *"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, H).
6. *"Мультифрактальний аналіз детрендованих флуктуацій"*: MFDFA отримують шляхом розширення на q-й порядок узагальненого середньоквадратичного відхилення. Середньоквадратичне відхилення q-го порядку може розрізняти сегменти з малими та великими флуктуаціями. Степенева залежність між середньоквадратичним відхиленням q-го порядку чисельно визначається як узагальнейний показник Херста q-го порядку.
7. *"Мультифрактальний спектр часових рядів"*: на основі показника Херста q-го порядку обчислено декілька мультифрактальних спектрів.
8. *"Узагальнені фрактальні розмірності"* представляє більш детальний опис показників D_q, що будуть описані в подальшому.
9. *"Аналогії мультифракталів із термодинамікою"* показує, що отримані кількісні мультифрактальні показники мають зв'язок із термодинамічними показниками, що дозволило нам вивести мультифрактальну "теплоємність".

Для подальшої візуалізації кожного кроку процедури MFDFA імпортуємо наступні модулі:

```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
```

```
import numpy as np
import neurokit2 as nk
import yfinance as yf
import pandas as pd
import scienceplots
from scipy.integrate import cumulative_trapezoid
from tqdm import tqdm

%matplotlib inline
```

І виконаємо налаштування рисунків для виведення:

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                   # розмір фонтів рисунку
'lines.linewidth': 2,                # товщина ліній
'axes.titlesize': 'small',           # розмір титулки над рисунком
'axes.labelsize': size,              # розмір підписів по осям
'legend.fontsize': size,             # розмір легенди
'xtick.labelsize': size,             # розмір розмітки по осі Ox
'ytick.labelsize': size,             # розмір розмітки по осі Ox
"font.family": "Serif",              # сімейство стилів підписів
"font.serif": ["Times New Roman"],   # стиль підпису
'savefig.dpi': 300,                  # якість збережених зображень
'axes.grid': False                   # побудова сітки на самому рисунку
}

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 згенерованими монофрактальними рядами, складність яких, очевидно, має бути меншою.

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 (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' # символ 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 Ox
ylabel = symbol          # підпис по вісі Oy
```

Виведемо досліджуваний ряд:

```
fig, ax = plt.subplots(1, 1)                # 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 Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

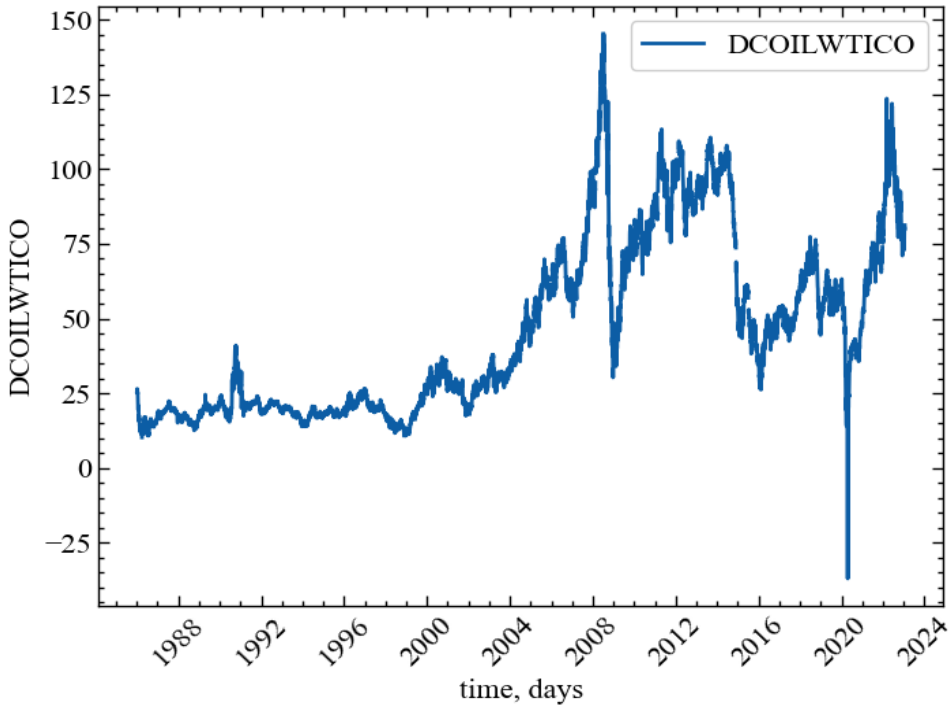


Рис. 7.5: Динаміка щоденних змін індексу сирої нафти WTI

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 Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}_filtered.jpg')       # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

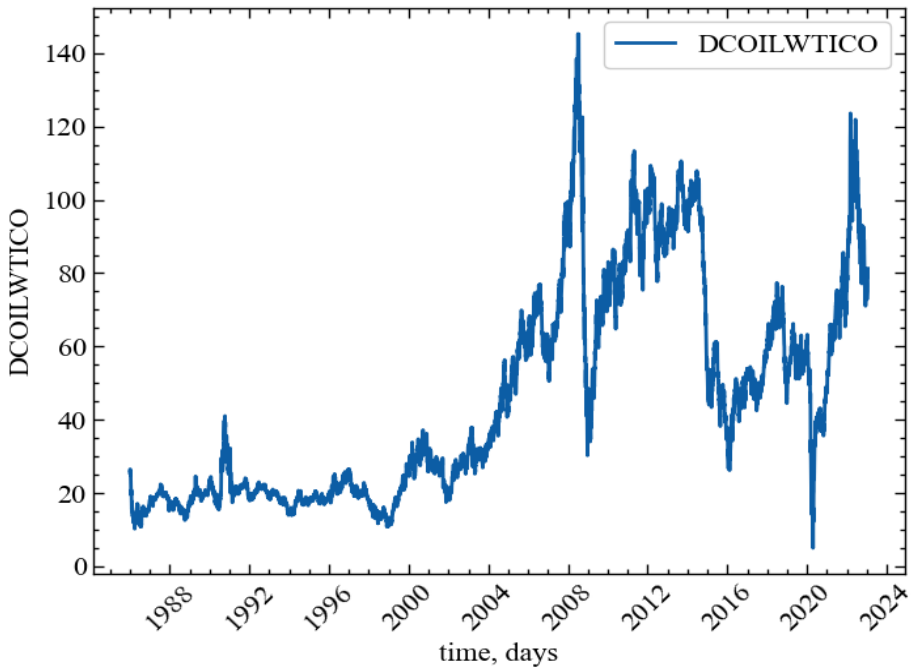


Рис. 7.6: Динаміка щоденних змін індексу сирої нафти WTI (із видаленими NaN та заміненим від'ємним значенням)

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: $\text{beta} = -2$;
- синій шум: $\text{beta} = -1$;

- білий шум: beta = 0;
- флікер/рожевий шум: beta = 1;
- коричневий шум: beta = 2.

Її синтаксис:

```
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)
```

7.1.4.1 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 [114]. 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, наведеного на Рис.7.7:

```
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 для нафти

fig, ax = plt.subplots(3, 1, sharex=True)

ax[0].plot(time_ser.index[1:], wti_ret)
ax[0].plot(time_ser.index[1:], RW3, 'r')
ax[0].margins(x=0)
ax[0].set_title('Мультифрактальний часовий ряд', fontsize=16)

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].margins(x=0)
ax[1].set_title('Монофрактальний часовий ряд', fontsize=16)
ax[1].legend()

ax[2].plot(time_ser.index[1:], white_noise)
ax[2].plot(time_ser.index[1:], RW1, 'r')
ax[2].margins(x=0)
ax[2].set_title('Білий шум', fontsize=16)

plt.show();
```

Рис. 7.7: Мультифрактальний (верхня панель), монофрактальний (середня панель) та подібний до білого шуму (нижня панель) часові ряди

7.1.4.2 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, 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].margins(x=0)
ax[0].set_title('Мультифрактальний часовий ряд', fontsize=16)

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].margins(x=0)
ax[1].set_title('Монофрактальний часовий ряд', fontsize=16)

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].margins(x=0)
ax[2].set_title('Білий шум', fontsize=16)

handles, labels = ax[0].get_legend_handles_labels()
fig.legend(handles, labels, loc='lower center')

plt.show();
```

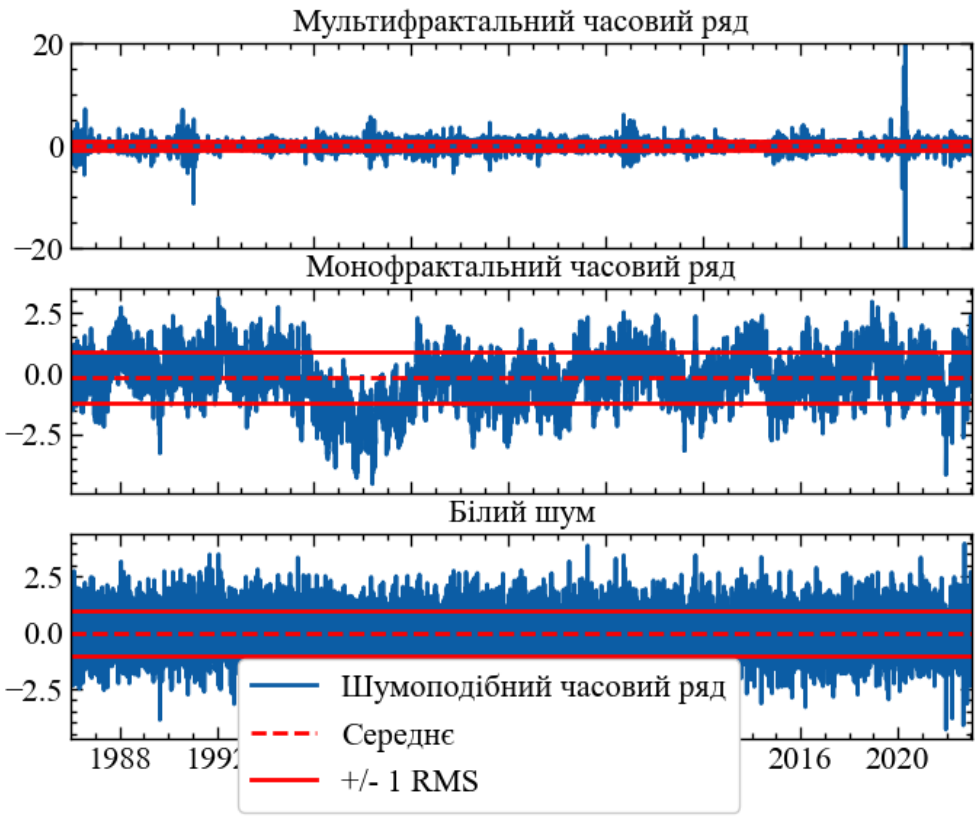


Рис. 7.8: Мультифрактальний (верхня панель), монофрактальний (середня панель) та подібний до білого шуму (нижня панель) часові ряди з нульовим середнім значенням (червона штрихова лінія) і ±RMS (червона суцільна лінія)

Рис. 7.8 ілюструє, що середня амплітуда варіації (тобто середньоквадратичне відхилення) є однаковою для всіх трьох часових рядів, навіть якщо вони мають досить різну структуру. MFDFA може розрізняти ці структури.

7.1.4.3 Локальна середньоквадратична варіація часового ряду

Мультифрактальні часові ряди на верхній панелі мають локальні флуктуації різної величини. Середньоквадратичне відхилення (RMS) у попередньому коді можна обчислити для сегментів часового ряду, щоб розрізнити величини локальних флуктуацій. Проста процедура полягає в тому, щоб розділити 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:

```
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 X. 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 X, де `len(X)` — розмір вибірки часового ряду X. Таким чином, `segments = 6` при `len(X) = 9335` і `scale = 1335`. З дев'ятого по шістнадцятий рядки — це цикл, що 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 $v = 1$, `Index[1]` переходить від 1335 до 2670 значення другого

сегмента. В останньому циклі $v = 6$, `Index[6]` переходить від 8010-го до 9345-го значення (не включно).

7.1.4.4 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 X на кожному сегменті v підбирається поліноміальний тренд `fit[v]`. Параметр m визначає порядок полінома. Поліноміальний тренд є лінійним, якщо $m = 1$, квадратичним, якщо $m = 2$, і кубічним, якщо $m = 3$. 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 v. Локальна середньоквадратична варіація, `RMS[v]`, представлена на Рис. 7.9 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, 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].margins(x=0)
ax[0].set_title('Лінійне детрендування '+r'$(m=1)$', fontsize=16)


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].margins(x=0)
ax[1].set_title('Квадратичне детрендування '+r'$(m=2)$', fontsize=16)


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].margins(x=0)
ax[2].set_title('Кубічне детрендування '+r'$(m=3)$', fontsize=16)

handles, labels = ax[1].get_legend_handles_labels()
fig.legend(handles, labels, loc='lower center')

plt.show();
```

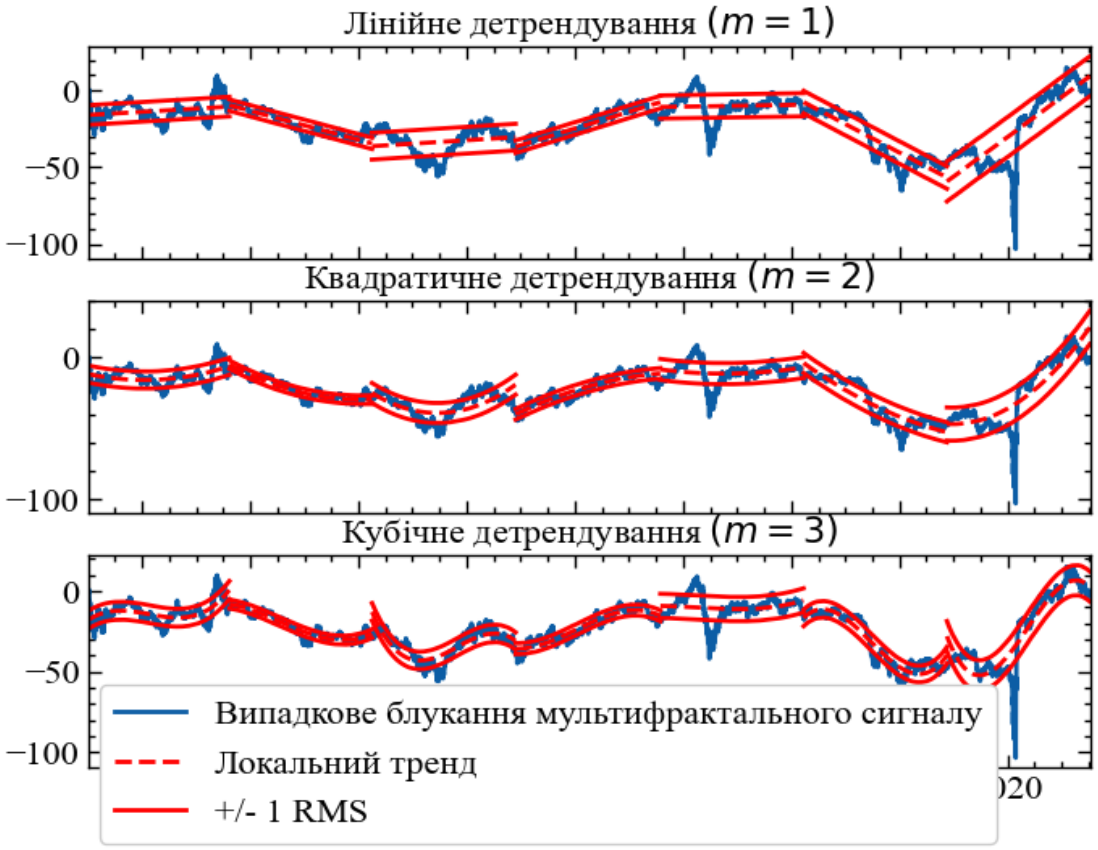


Рис. 7.9: Обчислення локальних флуктуацій `RMS` навколо лінійного, квадратичного та кубічного трендів за допомогою функції `calc_rms()` ($m = 1$, $m = 2$ та $m = 3$, відповідно). Червона пунктирна лінія — це підігнаний тренд, `fit[v]`, у семи сегментах вибірки розміром 1335. Відстань між червоним штриховим трендом і суцільними червоними лініями становить $\pm$`RMS`

7.1.4.5 Монофрактальний аналіз детрендованих флуктуацій

У DFA варіації локального середньоквадратичного відхилення кількісно оцінюються загальним середньоквадратичним відхиленням (F).

Швидкі коливання часового ряду X впливатимуть на загальне середньоквадратичне відхилення F у сегментах малої довжини (масштабу), тоді як повільні коливання впливатимуть на F у сегментах великої довжини (масштабу). Таким чином, функція флуктуацій F 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 $F(ns)$ 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), 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].set_title(f"Масштаб = {scales[scale]}", fontsize=16)
        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].set_title(f"Масштаб = {scales[scale]}", fontsize=16)
        ax[scale].margins(x=0)

handles, labels = ax[-1].get_legend_handles_labels()
fig.legend(handles, labels, loc='upper left', fontsize=14)

fig.tight_layout(pad=0.05)
plt.show();
```

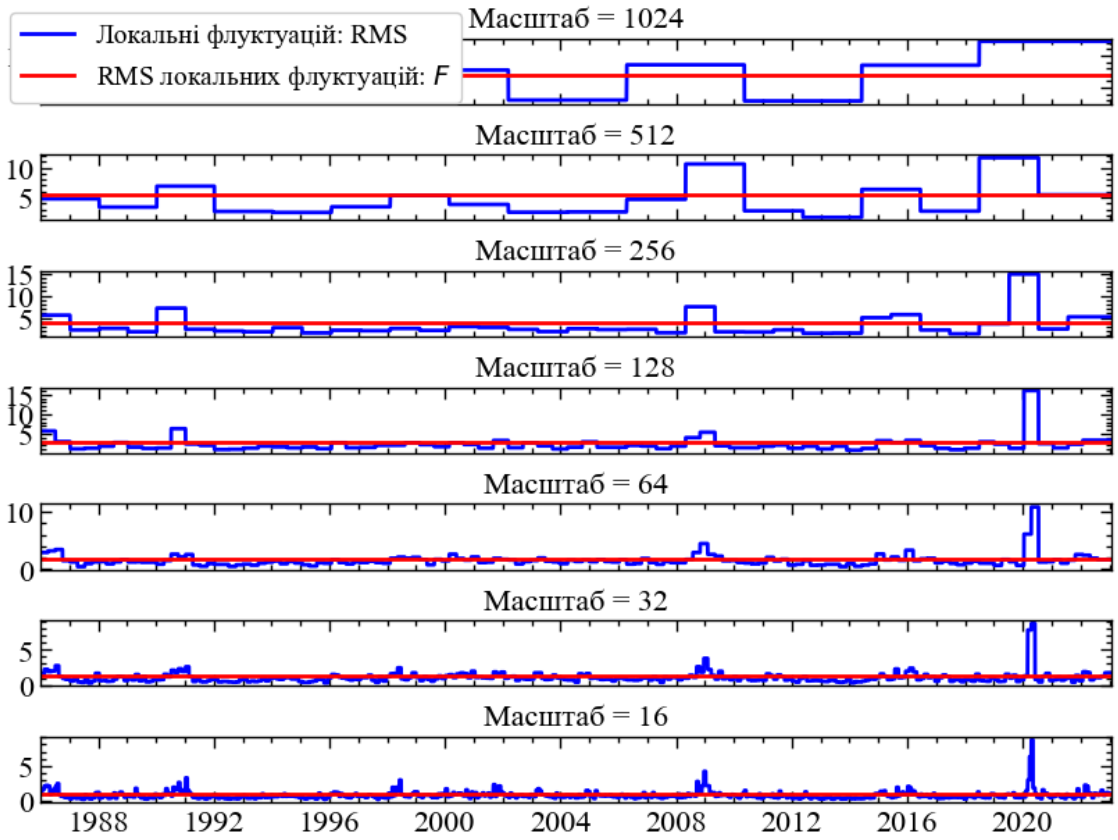


Рис. 7.10: Локальні флуктуації `RMS[ns]` обчислені для сегментів із різними масштабами. Функція флуктуацій `F[ns]` є загальним середньоквадратичним відхиленням локальних коливань `RMS[ns]`. Зверніть увагу, що `F[ns]` зменшується

зі зменшенням масштабу

DFA визначає монофрактальну структуру часового ряду відповідно до степеневої залежність між загальним середнім квадратичним відхиленням (тобто F), обчисленим для декількох масштабів. Степенева залежність між загальним середнім квадратичним відхиленням позначається нахилом (H) лінії регресії, розрахованим за допомогою наступного коду:

```
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 F 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=16)

fig.tight_layout()
plt.show();
```

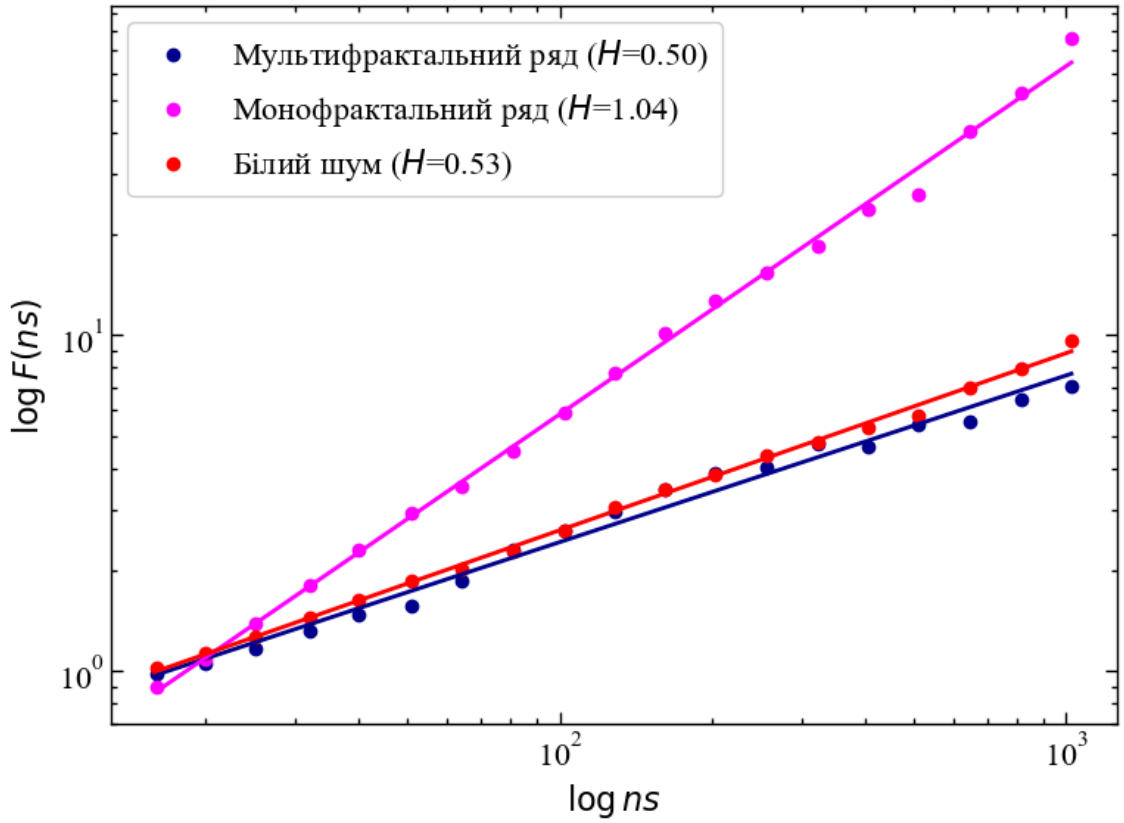


Рис. 7.11: Графік залежності загального середньоквадратичного відхилення (тобто, функції флуктуацій F) від масштабу. Масштабно-інваріантна залежність позначається нахилом H ліній регресії (показником Херста)

Показник Херста визначає монофрактальну структуру часового ряду, вказуючи, наскільки швидко зростає загальне середньоквадратичне відхилення F локальних коливань `RMS` зі збільшенням розміру локальних сегментів ряду (тобто, масштабу). Рис. 7.11 показує, що загальне середньоквадратичне значення локальних флуктуацій F у порівнянні з індексом нафти та білим шумом зростає швидше зі збільшенням розміру вибірки сегментів для монофрактального рожевого шуму. Рис. 7.12 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 зашумлений часових рядів, 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, 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].legend(loc="upper right", fontsize=12)
    ax[idx].margins(x=0)

fig.subplots_adjust(hspace=0)

plt.show();
```

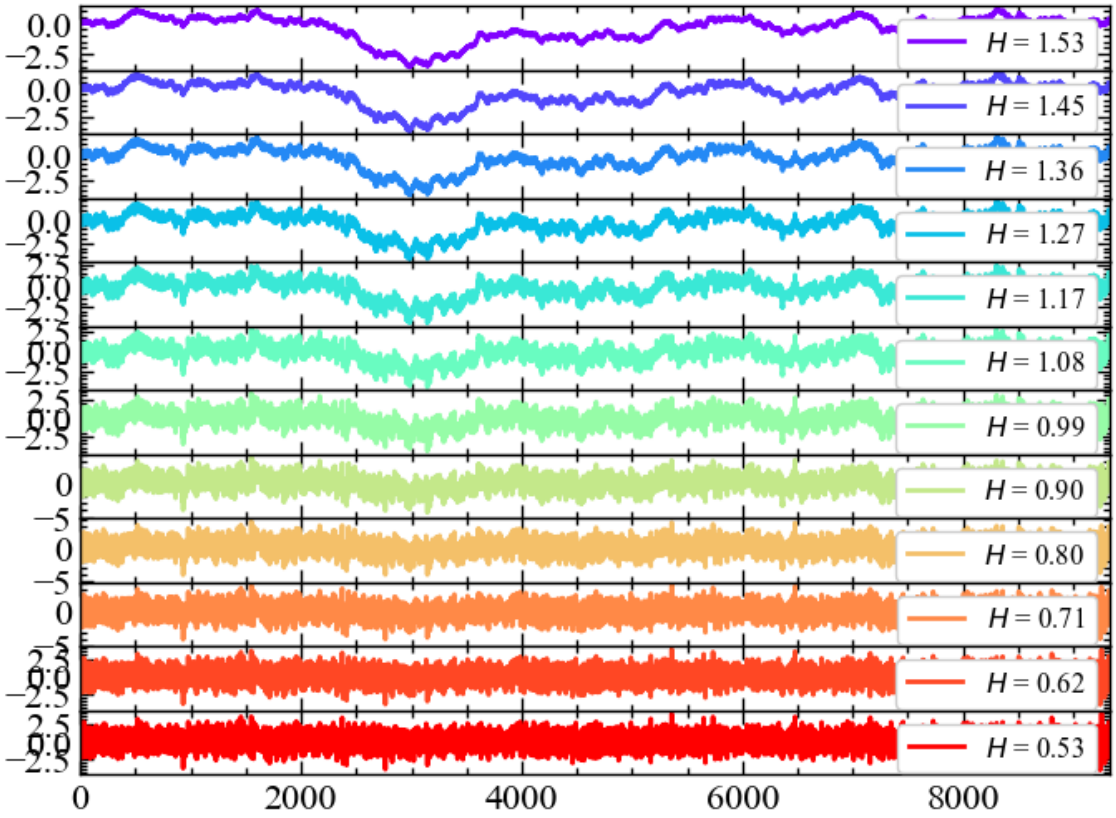


Рис. 7.12: Діапазон показників Херста визначає континуум фрактальних структур між білим шумом (H = 0.5) і коричневим шумом ($H = 1.5$). Рожевий шум $H = 1$ розділяє шуми $H < 1$, які мають більш помітні швидкі флуктуації, і випадкові блукання $H > 1$, які мають більш помітні повільні флуктуації

7.1.4.6 Мультифрактальний аналіз детрендованих флуктуацій

Структури монофрактального та мультифрактального часових рядів відрізняється, хоча вони мають схожі загальні середньоквадратичні значення. Мультифрактальні часові ряди містять локальні флуктуації як з екстремально малими, так і з екстремально великими значеннями, що не характерно для монофрактальних часових рядів. Відсутність флуктуацій з екстремально великими та малими значеннями призводить до нормального розподілу для монофрактального часового ряду, де варіація описується лише статистичним моментом другого порядку (дисперсією). Отже, монофрактальний DFA базується на статистиці другого порядку загального середньоквадратичного відхилення (тобто, F). У мультифрактальному часовому ряді локальні коливання, `RMS[ns][v]`, будуть екстремально великими для сегментів v в межах часових періодів великих коливань і екстремально малими для сегментів v в межах часових періодів малих коливань. Отже, мультифрактальні часові ряди не є нормально розподіленими і слід враховувати всі статистичні моменти q-го

порядку. Таким чином, необхідно розширити загальне середньоквадратичне значення монофрактального DFA до середньоквадратичної функції флуктуацій q-го порядку мультифрактального DFA (F_q):

```
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 q-порядку $F_q(nq)$ від від'ємних до додатних q. Порядок q зважує вплив сегментів ряду з великими та малими коливаннями, RMS, як показано на наступному рисунку. На величину $F_q(nq)$ для від'ємних q впливають сегменти v з малими RMS(v). Навпаки, на $F_q(nq)$ для додатних q впливають відрізки v з великими RMS(v). Локальні флуктуації RMS з великими та малими величинами класифікуються за величиною від'ємного або додатного порядку q відповідно. На F_q для $q = -3$ і 3 більше впливають відрізки v з найменшим і найбільшим RMS(v), відповідно, порівняно з F_q для $q = -1$ і 1. Середня точка $q = 0$ є нейтральною щодо впливу відрізків з малим та великим RMS. Зверніть увагу, що в останньому рядку коду нового блоку перевизначено окремий випадок $q(nq) = 0$, оскільки $1/0$ прямує до нескінченності (тобто, $1/q(q = 0) = \infty$). Читач також повинен помітити, що $F_q[q == 2]$ дорівнює статистиці другого порядку F, оскільки $\sqrt{x} = x^{1/2}$. 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, sharex=True)

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 inrange(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 inenumerate(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=14)
    ax[idx].plot(time_ser.index[1:], y)
    ax[idx].plot(time_ser.index[1:], y_pink)
    ax[idx].margins(x=0)

handles, labels = ax[0].get_legend_handles_labels()

fig.tight_layout(pad=0.01)
plt.show();
```

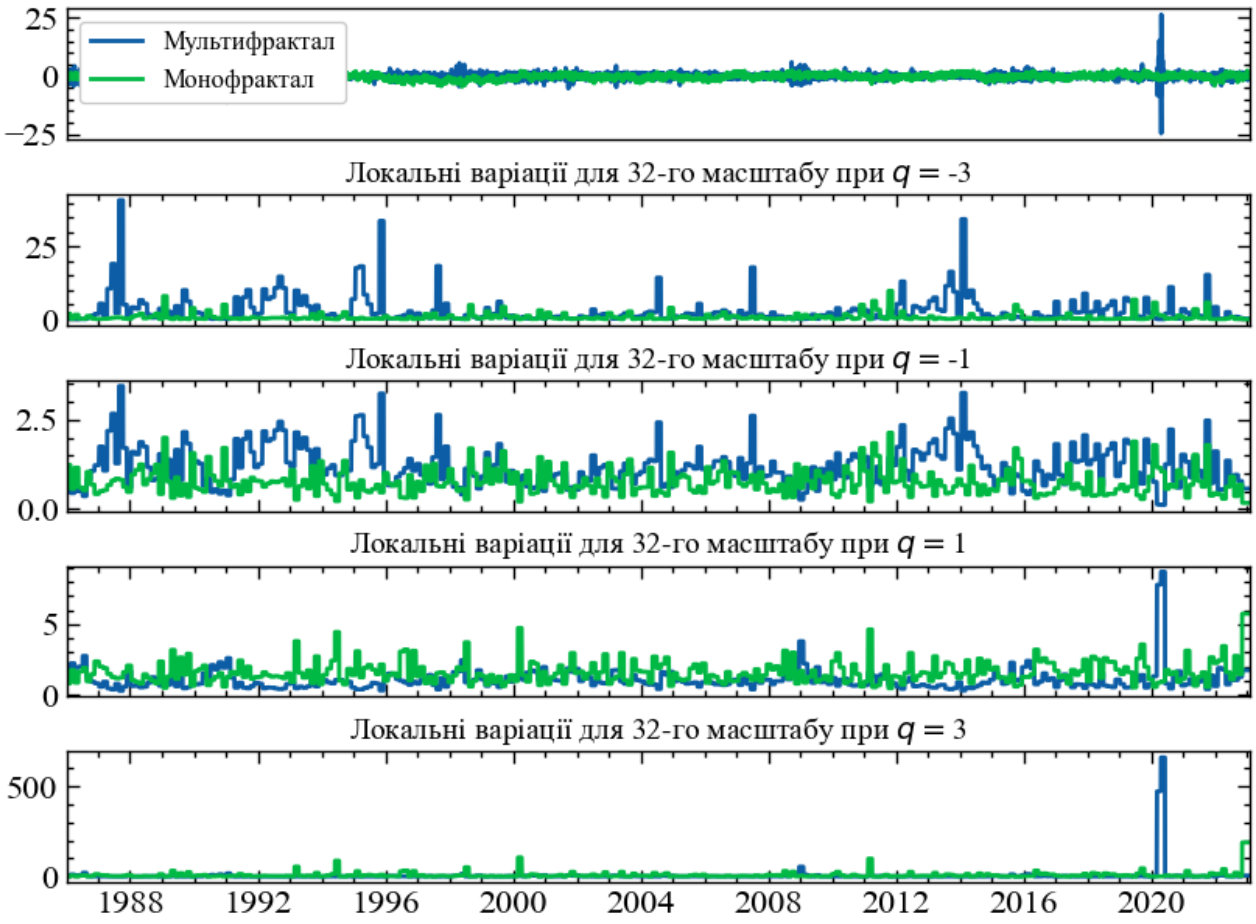


Рис. 7.13: Ілюстрація залежності локальних флуктуацій qRMS від q при масштабі 32

qRMS на Рис. 7.13 — це q-порядок локальних флуктуацій (тобто, RMS) і є складовою частиною загальногоq-порядку RMS (тобто, F_q). qRMS представлено для монофрактального (зелена смуга) та мультифрактальних (синя смуга) часових

рядів. Від'ємний порядок q ($q = -3$ і -1) підсилює сегменти в мультифрактальному часовому ряді з екстремально малими RMS, тоді як додатний порядок q ($q = 3$ і 1) підсилює відрізки з екстремально великими RMS. Зверніть увагу, що $q = -3$ і $q = 3$ підсилюють малу і велику варіацію відповідно більше, ніж $q = -1$ і $q = 1$. 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 q-го порядку здатне розрізняти структуру малих і великих флуктуацій і, відповідно, монофрактальних і мультифрактальних часових рядів.

Тепер можна визначити показники Херста q-го порядку як нахили $h(q)$ ліній регресії для кожного середньоквадратичного значення F_q q-го порядку. І $h(q)$, і лінія регресії визначаються в циклі для кожного q-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 inrange(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):
            # якщо флуктуації дорів. 0, log2 стикнеться з діленням на 0
            old_setting = np.seterr(divide="ignore", invalid="ignore")
            C = np.polyfit(np.log(scale), np.log(Fq[nq, :]), m)
            np.seterr(**old_setting)
            hq[nq] = C[0]
            qRegLine[nq] = np.polyval(C, np.log(scale))
        # ----------------------------

    return hq, qRegLine, Fq

scmin = 16
scmax = 1024
scres = 19

q_min = -5.0
q_max = 5.0
q_step = 0.1

nq = np.arange(q_min, q_max+q_step, q_step)

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)

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(loc='center right', fontsize=12)

fig.tight_layout(pad=0.1)
plt.show();
```

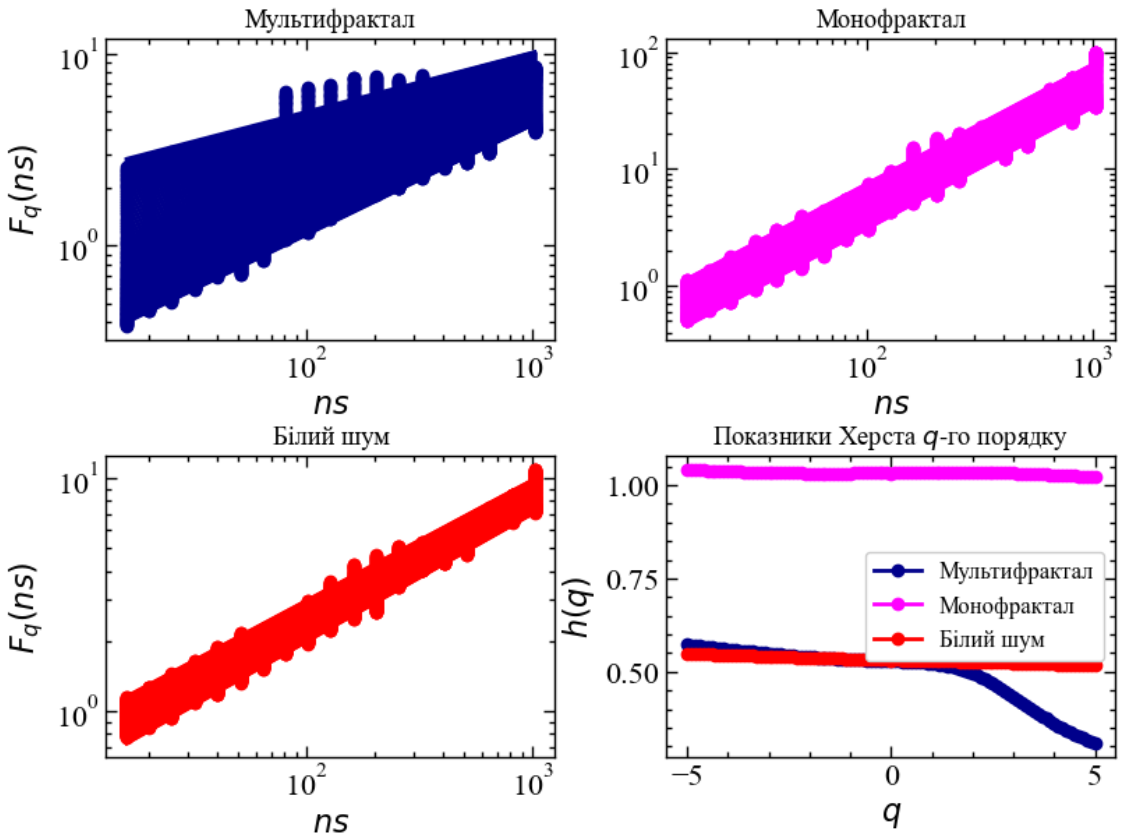


Рис. 7.14: Середньоквадратичні значення F_q для різних q-их порядків та відповідні лінії регресії, обчислені за допомогою MFDFA для мультифракталу, монофракталу та білого шуму

Можемо бачити, що узагальнена функція флуктуацій для мультифракталу залежить не лише від масштабу, але й від q, що демонструють різні нахили ліній регресії $h(q)$. Масштабуючі узагальнені функції флуктуацій F_q для монофракталу та білого шуму є q-незалежними, оскільки їх лінії регресії для різних масштабів мають один і той самий кут нахилу. Показник Херста q-го порядку $h(q)$ для мультифрактального ряду (синя лінія) представляється незалежним для $q < 0$ і змінним для $q > 0$. 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) $h(q)$ залишаються сталими.

7.1.4.7 Мультифрактальний спектр часових рядів

Показник Херста q-го порядку $h(q)$ 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 q-го порядку $\big(\tau(q)\big)$, а потім

через $\tau(q)$ отримати показник сингулярності q-го порядку $\big(\alpha(q)\big)$ і фрактальну розмірність $\big(f(\alpha)\big)$ флуктуацій (областей) із ступенем сингулярності $\alpha(q)$. Графік залежності $\alpha(q)$ від $f(\alpha)$ 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)

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=0.01)
plt.show();
```

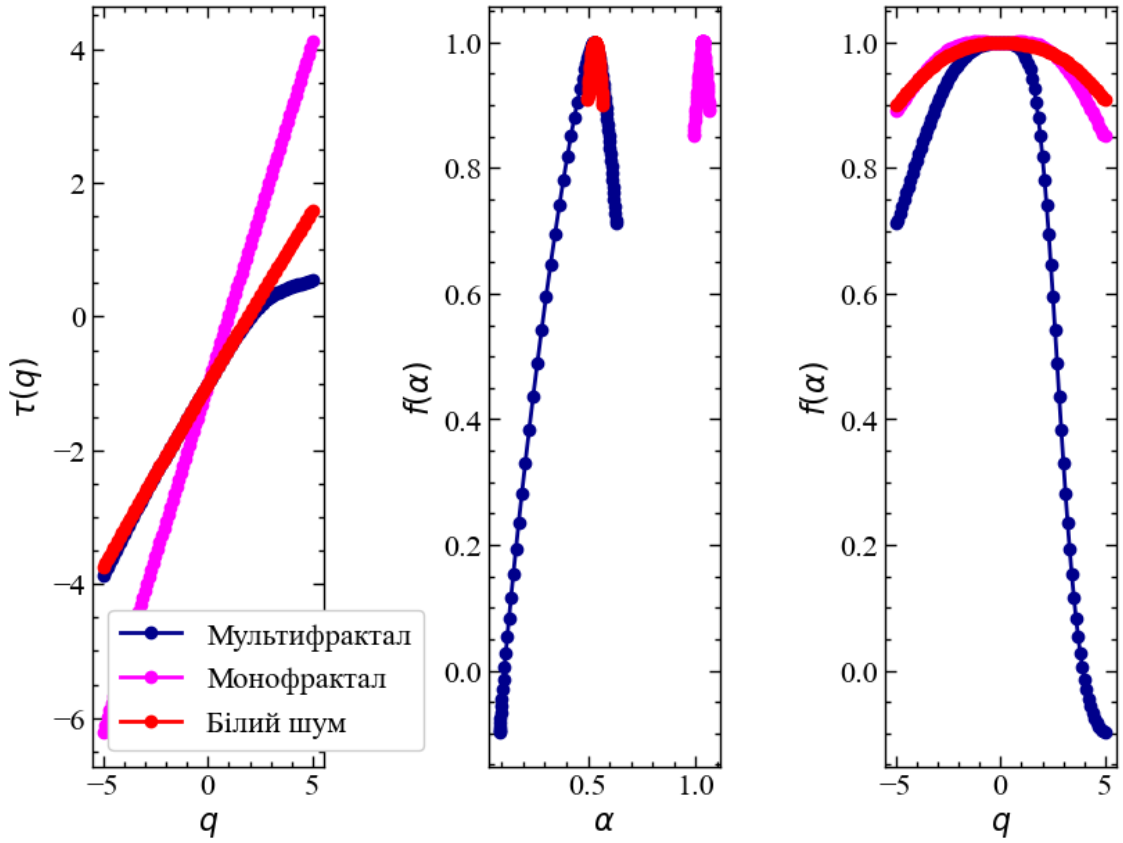


Рис. 7.15: Множинне представляння мультифрактального спектра для мультифрактала, монофрактала та білого шуму

Показники сингулярності α для великих висококонцентрованих флуктуацій малі й розташовані в лівому хвості спектра, тоді як α для малих флуктуацій великі й розташовані в правому хвості спектра.

Таким чином, сила мультифрактальності описується великим відхиленням експоненти локальної сингулярності α від центральної тенденції $\alpha(0)$. Монофрактальний сигнал — це випадок, коли α залишається майже константною, і в деяких випадках мультифрактальний спектр зводиться до однієї точки за даною α.

Діапазон α вказує на різноманітність експонент сингулярності, що описують динаміку системи, а величина $f(\alpha)$ вказує на величину внеску елементів із відповідним показником α.

Мультифрактальний спектр може характеризуватися різною шириною, що вказує на варіативність процесів, що відбуваються всередині системи. Так само він може бути як симетричним, так і асиметричним. Асиметрія буває як правосторонньою, так і лівосторонньою, що вказує на різний ступінь впливу висококонцентрованих і низькоконцентрованих елементів (флуктуацій). Мультифрактальний спектр матиме довгий лівий хвіст, коли часовий ряд має

мультифрактальну структуру, чутливу до локальних флуктуацій з великими амплітудами.

Навпаки, мультифрактальний спектр матиме довгий правий хвіст, коли він чутливий до локальних флуктуацій з малими амплітудами.

Проілюструємо залежність ширини спектра мультифрактальності від рівня флуктуацій у ряді. Дану залежність будемо демонструвати на прикладі рядів, що розподілятимуться згідно альфа-стабільному розподілу Леві, який ми ще розглядатимемо в наступних роботах. Для генерації випадкових величин із даного розподілу, використовуватимемо модуль `scipy.stats`. З нього імпортуємо клас `levy_stable` для використання методу `rvs()`. Метод приймає показник α, що відповідає за частоту виникнення подій, що виходять за межі нормального розподілу. Розглянемо діапазон таких значень α та спектри згенерованих рядів.

```
from scipy.stats import levy_stable

alphas = np.linspace(1.5, 2.0, 7)
scmin = 16
scmax = 1024
scres = 19

q_min = -5.0
q_max = 5.0
q_step = 0.1
nq_levy = np.arange(q_min, q_max+q_step, q_step)

exponents = np.linspace(np.log(scmin), np.log(scmax), scres)
scales_exp = np.round(np.exp(1)**exponents).astype(int)

color = iter(plt.cm.plasma(np.linspace(0, 0.8, len(alphas))))

fig = plt.figure()
subfigs = fig.subfigures(1, 2)
ax1 = subfigs[0].subplots(len(alphas), 1, sharex=True)
ax2 = subfigs[1].subplots(1, 1)

for i in range(len(alphas)):

# генеруємо альфа-стабільний процес
    r = levy_stable.rvs(alpha=alphas[i], beta=0, loc=0,
                        scale=1, size=len(wti_ret), random_state=123)

    Hq_levy, qRegLine_levy, Fq_levy = calc_Hq(r, scale=scales_exp, q=nq_levy,
m=1)
    tau_levy = nq_levy * Hq_levy - 1
    alpha_levy = np.gradient(tau_levy, nq_levy)
    f_levy = nq_levy * alpha_levy - tau_levy

    c = next(color)
    ax1[i].plot(np.arange(len(r)), r, label=fr'$\alpha$={alphas[i]:.2f}', c=c)
    ax1[i].margins(x=0)
```

```
    ax1[i].legend(loc="upper left", fontsize=12)
    ax2.plot(alpha_levy, f_levy, marker='o', c=c)

ax1[0].set_title("Мультифрактальні часові ряди", fontsize=16)
ax1[-1].set_xlabel("Час (порядковий номер)")
ax1[len(alphas)//2].set_ylabel('Амплітуда коливань')

ax2.set_title("Мультифрактальні спектри", fontsize=16)
ax2.set_xlabel(r"$\alpha$")
ax2.set_ylabel(r"$f(\alpha)$")

fig.subplots_adjust(hspace=0.1)

plt.show();
```

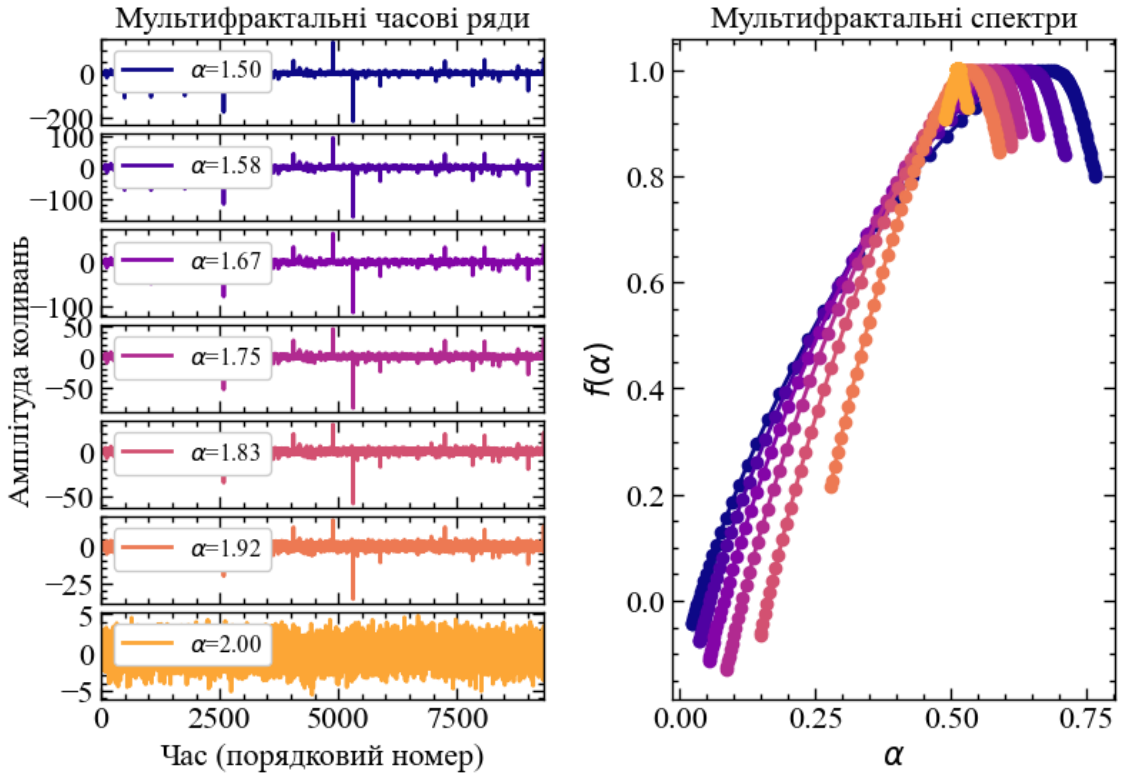


Рис. 7.16: Ілюстрація множини мультифрактальних часових рядів (Леві альфа-стабільних процесів) та їх мультифрактальних спектрів, що були згенеровані з різними значеннями α. Зверніть увагу на зростання структурних відмінностей між періодами з малими і великими флуктуаціями зі збільшенням ширини мультифрактального спектра

Система, складність якої зумовлена висококонцентрованими елементами, матиме чітко виражений лівосторонній спектр. Складність системи, що зумовлена слабко концентрованими елементами, характеризує правий хвіст мультифрактального спектра. Якщо складність системи розвивається за рахунок елементів двох типів, тоді спектр представлятиметься симетричним, де елементи двох типів будуть рівноймовірними. Для згенерованих вище Леві альфа-

стабільних процесів видно, що чим нижче значення α, тим сильніша домінація висококонцентрованих (великих) флуктуацій. При $\alpha = 2.0$ спектр усе сильніше звужується до сингулярної точки.

Далі буде показано, що для отриманої мультифрактальної параболи мультифрактального спектра можуть бути розраховані показники як **всієї ширини спектра** ($\Delta\alpha$), так і окремо його **правого** та **лівого хвостів** (R та L). Також можна розрахувати **значення сингулярності**, де $f(\alpha)$ приймає максимальне значення α_0, і навіть так звану "**асиметрію**" цього спектра (Δf). На Рис. 7.17 схематично представлено положення ключових індикаторів мультифрактальності спектра.

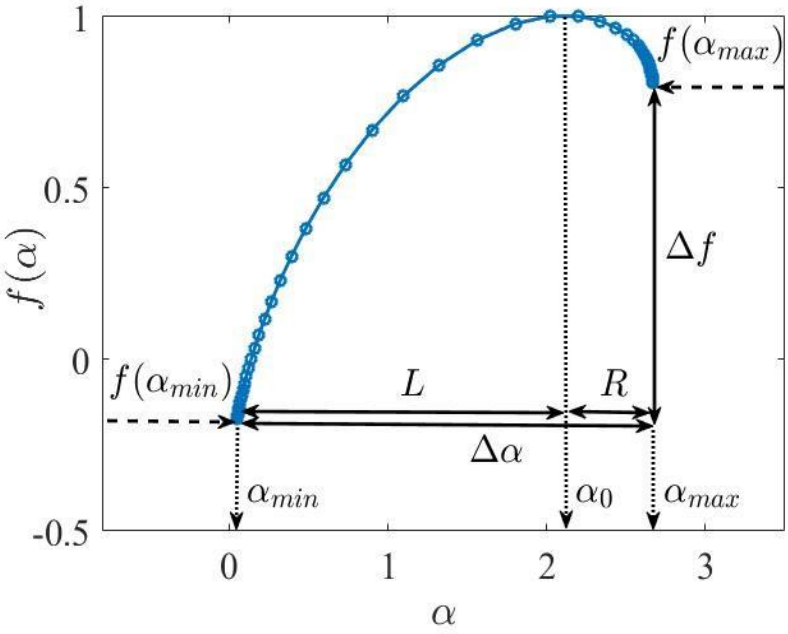


Рис. 7.17: Графік мультифрактального спектра із відміченними на ньому показниками ширини спектра мультифрактальності ($\Delta\alpha$), значень мінімальної, центральної та максимальної сингулярності ($\alpha_{min}, \alpha_0, \alpha_{max}$), ширини лівого та правого хвостів спектра (L, R) та різниці між фрактальними розмірностями в кінцях параболи (Δf)

Також варто зазначити, що дана схема не представляє вичерпний список індикаторів мультифрактальності системи, що ми використовуватимемо в подальшому, але має надавати інтуїтивне розуміння того, як виводиться більшість мультифрактальних показників.

7.1.4.8 Узагальнені фрактальні розмірності

Наряду з мультифрактальним спектром корисно буде розглянути спектр узагальнених фрактальних розмірностей або по іншому, **розмірностей Реньї**

(Renyi dimensions), оскільки вони також мають інформаційно-теоретичне значення. З'ясуємо фізичний сенс узагальнених фрактальних розмірностей для деяких значень q. При $q = 0$, $Z(0, \varepsilon) = N(\varepsilon)$. З іншого боку, можна визначити, що $Z(0, \varepsilon) \approx \varepsilon^{\tau(0)} = \varepsilon^{-D_0}$. Співставляючи зазначені рівності, можемо прийти до співвідношення $N(\varepsilon) \approx \varepsilon^{-D_0}$. Отже, величина D_0 представляє собою звичайну хаусдорфову розмірність множини Ω. Також вона відповідає максимуму мультифрактального спектра, $f(\alpha)$, що завжди дорівнює одиниці для одновимірного сигналу. Для задач розпізнавання кризових явищ ця характеристика є найгрубішою і не несе інформації про статистичні властивості системи.

Тепер з'ясуємо сенс величини D_1. Оскільки при $q = 1$ статистична сума $Z(1, \varepsilon) = 1$, то $\tau(1) = 0$. Таким чином, ми маємо невизначеність, коли $D_1 = \tau(1)/(1 - 1)$. Розкриємо цю невизначеність за допомогою наступної рівності:

$$Z(q, \varepsilon) = \sum_{i=1}^{N(\varepsilon)} p_i^q = \sum_{i=1}^{N(\varepsilon)} p_i \exp[(q - 1) \ln p_i].$$

Тепер, спрямовуючи $q \to 1$, розкладаючи експоненту і враховуючи умову нормування ймовірностей p_i, отримуємо

$$Z(q \to 1, \varepsilon) \approx \sum_{i=1}^{N(\varepsilon)} [p_i + (q - 1) p_i \ln p_i] = 1 + (q - 1) \sum_{i=1}^{N(\varepsilon)} p_i \ln p_i.$$

У результаті ми приходимо до наступного виразу:

$$D_1 = \lim_{\varepsilon \to 0} \sum_{i=1}^{N(\varepsilon)} p_i \ln p_i \Big/ \ln \varepsilon.$$

З точністю до знака, чисельник у цій формулі представляє собою **інформаційну ентропію** (information entropy) фрактальної множини $S(\varepsilon)$:

$$S(\varepsilon) = -\sum_{i=1}^{N(\varepsilon)} p_i \ln p_i.$$

Таким чином, результуюча величина узагальненої фрактальної розмірності D_1 пов'язана з ентропією $S(\varepsilon)$ наступним співвідношенням:

$$D_1 = -\lim_{\varepsilon \to 0} S(\varepsilon)/\ln \varepsilon.$$

Повертаючись до задачі розподілу точок на фрактальній множині Ω, можно сказати, що оскільки $S(\varepsilon) \approx \varepsilon^{-D_1}$, величина D_1 характеризує інформацію, необхідну для опису положення точки в деякій комірці.

Додаткова інформація по інформаційній розмірності

Інформаційна розмірність може бути використана для опису просторової неоднорідності системи. Що однорідніший атрактор, то вищим має бути цей показник. Тобто, чим більшу кількість конфігурацій здатні займати елементи даної системи, тим більша кількість інформації нам потрібна для обліку кожного елемента. За просторової однорідності інформаційна ентропія так само зростає, що пов'язує інформаційну розмірність із поняттям ентропії. Оскільки D_1 це тангенс кута нахилу лінії регресії, що будується для залежності між ентропією та радіусом кіл, у яких вимірюється частота влучання окремих елементів атрактора, можна сказати, що інформаційна розмірність відображає швидкість зміни інформаційної ентропії. Чим вища D_1, тим стрімкіше зростає ентропія — міра нашого нинішнього незнання про систему. Чим нижча D_1, тим менша сама ентропія. Інакше кажучи, тим більша просторова асиметрія, упорядкованіша складність, вищі наші знання про поточний стан системи, і тим менше інформації нам потрібно для опису тих конфігурацій, які система може займати

Для узагальненої фрактальної розмірності при $q = 2$ справедливий наступний вираз:

$$D_2 = \lim_{\varepsilon \to 0} \ln \sum_{i=1}^{N(\varepsilon)} p_i^2 \Big/ \ln \varepsilon.$$

Величина p_i представляє собою ймовірність попадання точки в комірку розміром ε. Тоді величина p_i^2 є ймовірністю попадання в цю комірку двох точок. Знаходячи суму p_i^2 по всім зайнятим коміркам, ми отримуємо ймовірність того, що дві навмання обрані точки з множини Ω знаходяться всередині однієї комірки з розміром ε. Отже, відстань між цима двома точками буде менше або порядку ε. Ймовірність знаходження двох траєкторій у межах околиці з радіусом ε можна знайти за допомогою кореляційного інтеграла.

У такому разі ми приходимо до висновку, що узагальнена розмірність визначає залежність кореляційного інтеграла $C(\varepsilon)$ від ε. З цієї причини величину D_2 в літературі іменують **кореляційною розмірністю** (correlation dimension).

Тепер проаналізуємо поведінку $f(\alpha)$. Значення функції в максимумі легко визначити, якщо скористатися виразом (7.7), де $\tau(q) = q\alpha(q) - f\big(\alpha(q)\big)$ або $(q-1)D_q = q\alpha(q) - f\big(\alpha(q)\big)$. При $q = 0$ ми отримаємо, що $f(\alpha_0) = D_0$, тобто максимальне значення спектра дорівнює хаусдорфовій розмірності.

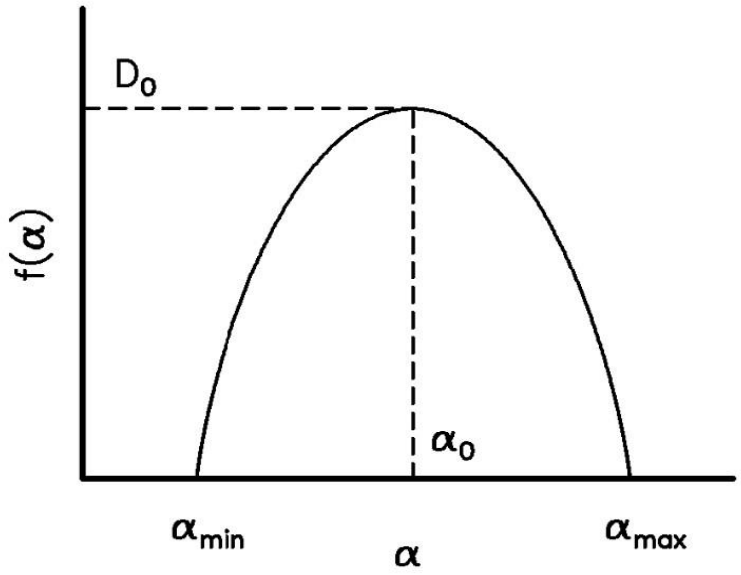


Рис. 7.18: Максимум функції $f(\alpha)$ дорівнює фрактальній розмірності D_0

Розглянемо випадок, коли $q = 1$. Оскільки $\tau(1) = 0$, тоді з рівняння вище слідує, що $\alpha(1) = f\big(\alpha(1)\big)$. З іншого боку ми знаємо, що оскільки $q = df(\alpha)/d\alpha$, похідна $f(\alpha)$ в цій точці дорівнює 1. Диференціюючи співвідношення $\tau(q) = (q-1)D_q$ по q,

$$\frac{d\tau}{dq} = D_q + (q-1)D_q' = \alpha(q),$$

і припускаючи, що $q = 1$, ми отримуємо, що $\alpha(1) = D_1$. Таким чином, ми маємо $D_1 = \alpha(1) = f\big(\alpha(1)\big)$. Отже, інформаційна розмірність D_1 лежить на кривій $f(\alpha)$ в точці, де $\alpha = f(\alpha)$ і $f'(\alpha) = 1$.

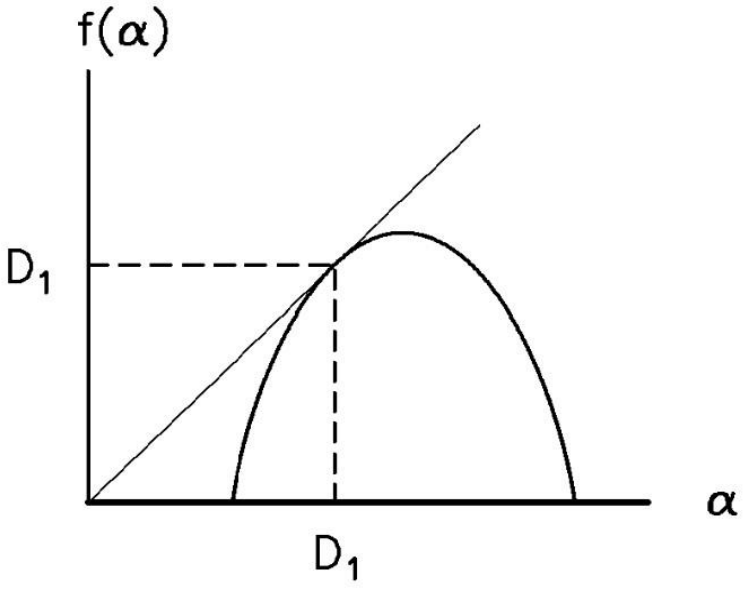


Рис. 7.19: Положення інформаційної розмірності D_1: $D_1 = \alpha = f(\alpha)$

Тепер розглянемо випадок, коли $q = 2$. Користуючись попередньою формулою, отримаємо, що $D_2 = 2\alpha(2) - f\big(\alpha(2)\big)$ або $f\big(\alpha(2)\big) = 2\alpha(2) - D_2$.

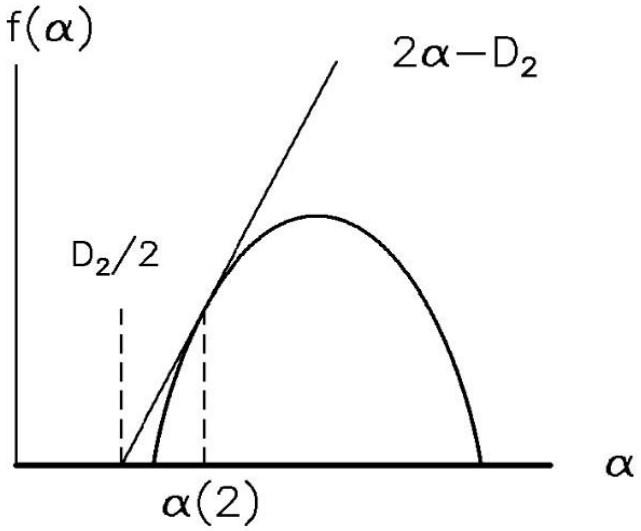


Рис. 7.20: Геометричне визначення кореляційної розмірності D_2

Далі розглянемо залежність узагальненої фрактальної розмірності D_q від різних значень q для мультифрактального ряду, монофрактального та білого шуму.

```
difference_zero = np.absolute(nq-0)
idx_zero = difference_zero.argmin()

difference_one = np.absolute(nq-1)
idx_one = difference_one.argmin()
```

```
difference_two = np.absolute(nq-2)
idx_two = difference_two.argmin()

# ініціалізуємо масиви під розмірності
Dq_multifrac = np.zeros(len(nq))
Dq_monofrac = np.zeros(len(nq))
Dq_white_noise = np.zeros(len(nq))

# Визначаємо узагальнені фрактальні розмірності там де q!=1
Dq_multifrac[nq!=nq[idx_one]] = tau_multifrac[nq!=nq[idx_one]] /
(nq[nq!=nq[idx_one]]-1)
Dq_monofrac[nq!=nq[idx_one]] = tau_monofrac[nq!=nq[idx_one]] /
(nq[nq!=nq[idx_one]]-1)
Dq_white_noise[nq!=nq[idx_one]] = tau_white_noise[nq!=nq[idx_one]] /
(nq[nq!=nq[idx_one]]-1)

# Визначаємо окремо узагальнені фрактальні розмірності при q=1
Dq_multifrac[nq==nq[idx_one]] = -tau_multifrac[nq==nq[idx_one]]
Dq_monofrac[nq==nq[idx_one]] = -tau_monofrac[nq==nq[idx_one]]
Dq_white_noise[nq==nq[idx_one]] = -tau_white_noise[nq==nq[idx_one]]

fig, ax = plt.subplots(1, 1)

ax.plot(nq, Dq_multifrac, linestyle='-', marker='o', label="Мультифрактал",
color='darkblue')
ax.plot(nq, Dq_monofrac, linestyle='-', marker='o', label="Монофрактал",
color='magenta')
ax.plot(nq, Dq_white_noise, linestyle='-', marker='o', label="Білий шум",
color='red')
ax.set_xlabel(r"$q$")
ax.set_ylabel(r"$D_{q}$")
ax.legend(loc="upper right")

ax.annotate(fr'$D_{0}$={Dq_multifrac[nq==nq[idx_zero]][0]:.2f}',
            xy=(nq[idx_zero], Dq_multifrac[nq==nq[idx_zero]]),
            xytext=(nq[idx_zero]-2, Dq_multifrac[nq==nq[idx_zero]]+2),
            arrowprops=dict(facecolor='black', shrink=0.05), fontsize=16)

ax.annotate(fr'$D_{1}$={Dq_multifrac[nq==nq[idx_one]][0]:.3f}',
            xy=(nq[idx_one], Dq_multifrac[nq==nq[idx_one]]),
            xytext=(nq[idx_one]-3, Dq_multifrac[nq==nq[idx_one]]-1.5),
            arrowprops=dict(facecolor='black', shrink=0.05), fontsize=16)

ax.annotate(fr'$D_{2}$={Dq_multifrac[nq==nq[idx_two]][0]:.2f}',
            xy=(nq[idx_two], Dq_multifrac[nq==nq[idx_two]]),
            xytext=(nq[idx_two], Dq_multifrac[nq==nq[idx_two]]-1.5),
            arrowprops=dict(facecolor='black', shrink=0.05), fontsize=16)

plt.show();
```

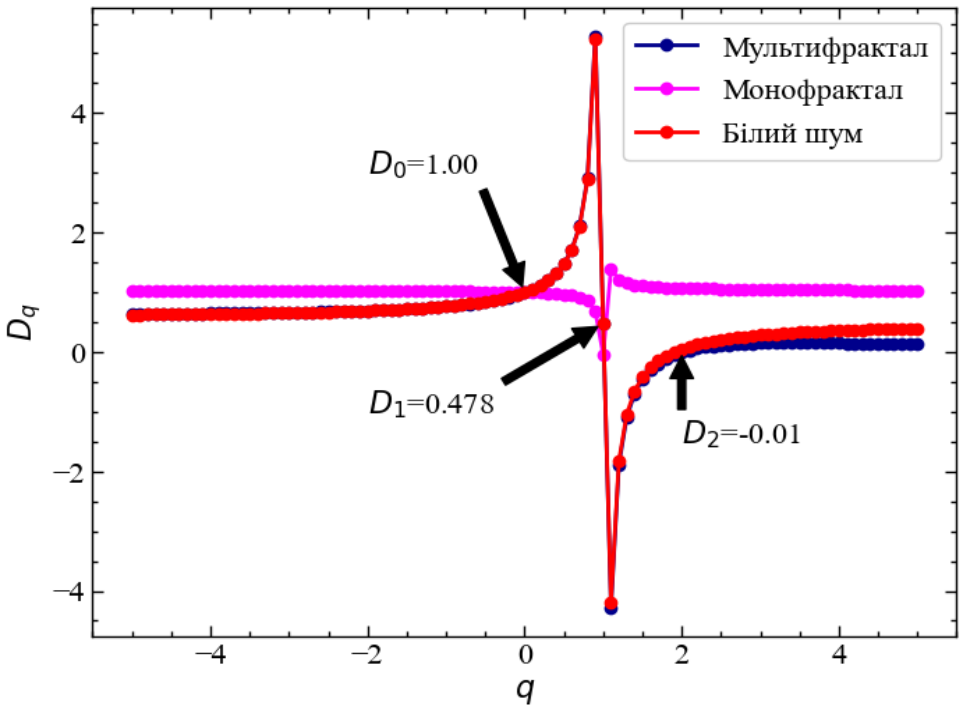


Рис. 7.21: Залежність узагальнених фрактальних розмірностей $D(q)$ від q

З представленого рисунку видно, що, по-переше, для всіх сигналів $D_0 = 1$, що відповідає теоретичним міркуванням. Інформаційна розмірність D_1 для мультифрактала та білого шуму однакова, що може говорити і про інформаційну зачущість обох сигналів. Для монофрактала вона близька до нуля. Кореляційна розмірність D_2 показує, що загалом як WTI, так і білий шум доволі схожі: їх значення в більшості своїй постають незалежними один від одного. Це розбігається з тими висновками, що були зробені в попередній роботі, де наближення D_2 до нуля вказувало на зростання ступеня корельованості системи. Для монофракталу D_2 знаходиться на рівні 1, що вказує на вищий ступінь кореляцій у цьому сигналі, порівнюючи з мульти- та монофракталами.

7.1.4.9 Аналогії мультифракталів із термодинамікою

Використовуючи концепції MFDFA, ми можемо по-новому поглянути на часовий сигнал як на термодинамічну систему. В рамках MFDFA показник маси $\tau(q)$ можна розглядати як аналоги вільної енергії, показник сингулярності α як аналог внутрішньої енергії U, мультифрактальний спектр $f(\alpha)$ як ентропію. Дійсно, форма мультифрактального спектра нагадує залежність ентропії термодинамічної системи від енергії U. Показники α_{min} та α_{max} можна охарактеризувати як верхній та нижній ліміти внутрішньої енергії системи. Функція $Z(q)$ є формальним аналогом статистичної суми $Z(\beta)$ в термодинаміці, де $\beta = 1/T = q$.

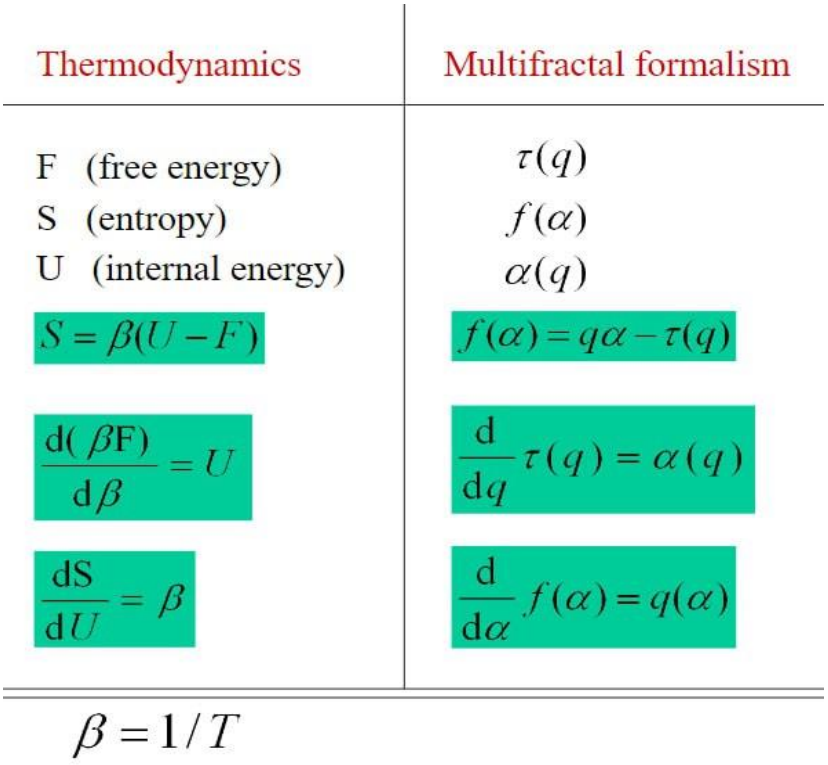


Рис. 7.22: Схематичне представлення аналогії мультифракталів із концепціями термодинаміки

Окрім цього, ми можемо прорахувати “температуру” досліджуваного сигнала. Більш конкретно, **мультифрактальну “питому теплоємність”** (multifractal heat capacity) $C(q)$ [115] можна визначити як

$$C(q) \equiv -(\partial^2\tau(q)/\,\partial q^2) = -(\partial\alpha/\,\partial q) \approx \tau(q+1) - 2\tau(q) + \tau(q-1).$$

Питома теплоємність, як міра швидкості зміни енергії, слугує індикатором явищ фазового переходу. У термодинамічній системі фаза характеризується однорідними фізичними властивостями, а фазовий перехід — стрибкоподібною зміною певних властивостей за критичної зовнішньої умови. Вивчення фазових переходів у мультифрактальному спектрі обмежувалося простими системами, такими як множина Кантора та логістична карта. Однак наш аналіз показує наявність фазових переходів [116] у мультифрактальному спектрі фінансових систем. Зокрема, “енергія” α виявляє значні флуктуації в околі q_c, які відображаються піком питомої теплоємності $C(q_c)$.

```
C_q_multifrac = -np.gradient(alpha_multifrac, nq, edge_order=2)
C_q_monofrac = -np.gradient(alpha_monofrac, nq, edge_order=2)
C_q_white_noise = -np.gradient(alpha_white_noise, nq, edge_order=2)

fig, ax = plt.subplots(1, 2)

ax[0].plot(nq, C_q_multifrac, linestyle='-', marker='o', label="Мультифрактал",
```

```
color='darkblue')
ax[0].plot(nq, C_q_monofrac, linestyle='-', marker='o', label="Монофрактал",
color='magenta')
ax[0].plot(nq, C_q_white_noise, linestyle='-', marker='o', label="Білий шум",
color='red')
ax[0].set_xlabel(r"$q$")
ax[0].set_ylabel(r"$C(q)$")
ax[0].legend(loc='center left')

ax[1].plot(alpha_multifrac, C_q_multifrac, linestyle='-', marker='o',
label="Мультифрактал", color='darkblue')
ax[1].plot(alpha_monofrac, C_q_monofrac, linestyle='-', marker='o',
label="Монофрактал", color='magenta')
ax[1].plot(alpha_white_noise, C_q_white_noise, linestyle='-', marker='o',
label="Білий шум", color='red')
ax[1].set_xlabel(r"$\alpha$")

fig.tight_layout(pad=0.3)
plt.show();
```

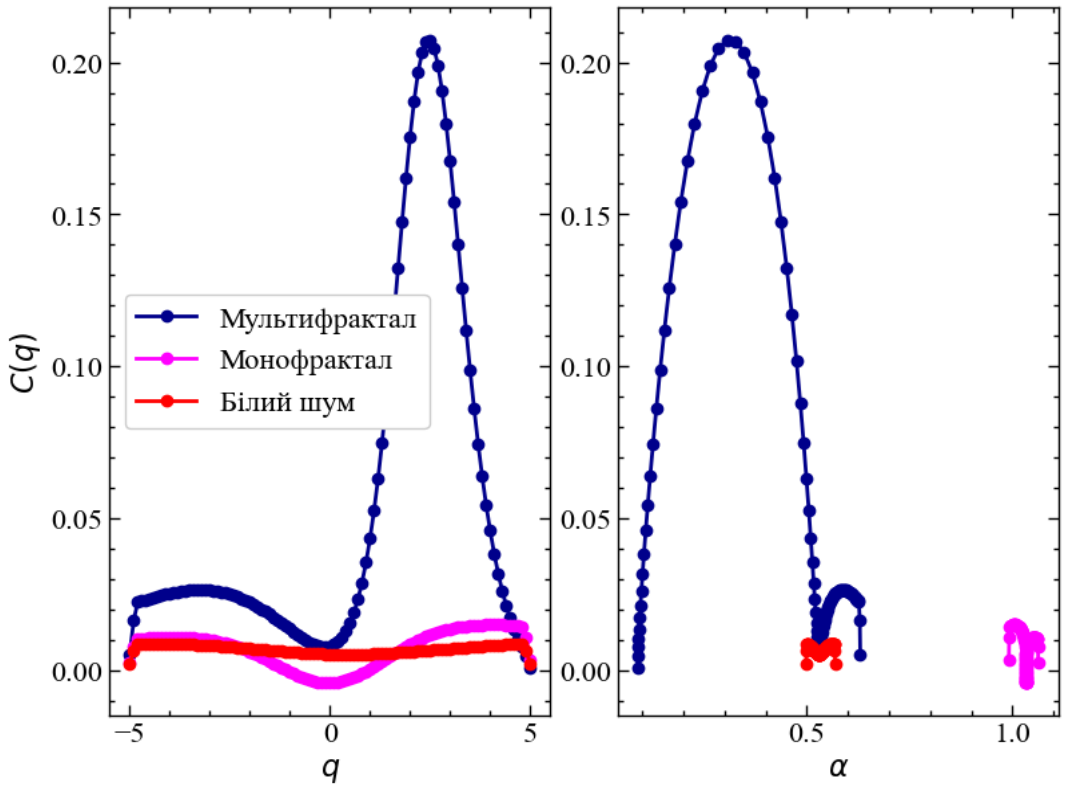


Рис. 7.23: Залежність мультифрактальної теплоємності $C(q)$ від q та α

Рис. 7.23 демонструє, що $C(q)$ досягає максимуму в діапазоні q від 2 до 3, що свідчить про те, що індекс WTI стає вкрай нерегулярним через велику флуктуацію катастрофи “COVID-19”, яка слугує квазіфазовим переходом досліджуваного індексу. Можна стверджувати, що колапс коронавірусної пандемії має найбільший вплив на мультифрактальність досліджуваного ряду.

7.2 Хід роботи

Звісно ж фрактальний аналіз усього ряду є важливим, але такий підхід ігнорує припущення про існування в часовій послідовності як монофрактальних, так і мультифрактальних ділянок. Тобто, ігнорується припущення про зміну ступеня складності з плином часу. Кількісні міри мультифрактальності, розраховані в рамках підходу ковзного вікна, є найбільш об'єктивними та практичними при аналізі систем. Окрім цього кількісні показники можуть бути використані як індикатори або індикатори-провісники аномальних явищ, або як базис для побудови іншої прогностичної моделі.

Повторно зчитаємо значення ряду із сайту FRED:

```
symbol = 'DCOILWTICO' # символ 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

#------- фільтрація від'ємних значень -------
time_ser = time_ser.dropna()    # видаляємо значення NaN
time_ser[time_ser.values<0] = 5 # замінюємо від'ємне значення на 5
#--------------------------------------------

xlabel = 'time, days'    # підпис по вісі Ox
ylabel = symbol          # підпис по вісі Oy
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance або FRED, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'           # підпис по вісі Ox
ylabel = symbol                     # підпис по вісі Oy
```

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 Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

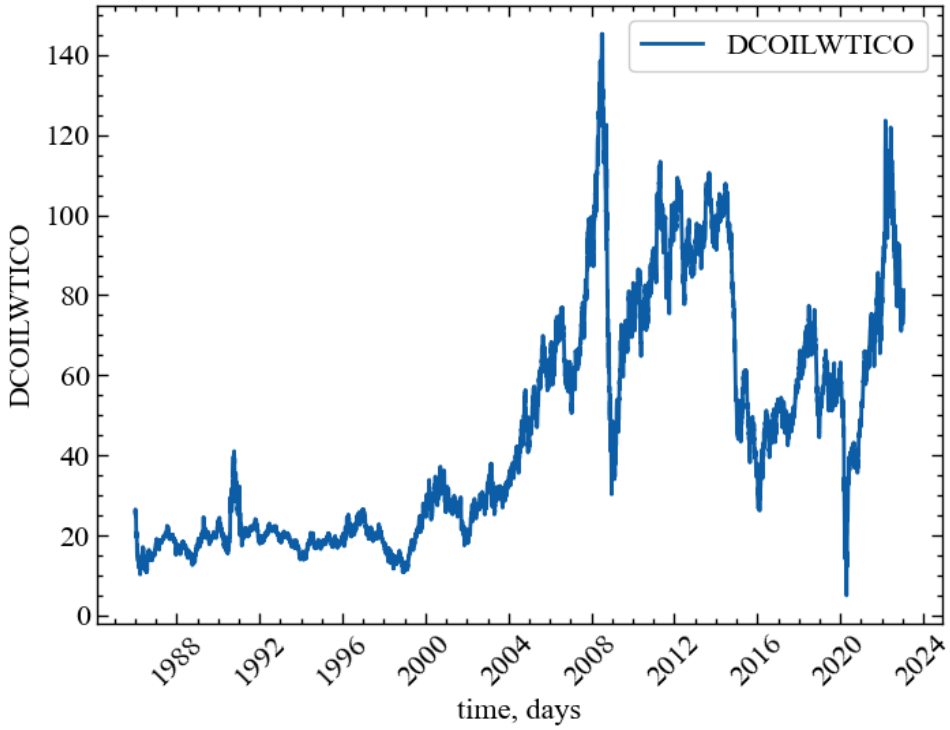


Рис. 7.24: Динаміка щоденних змін індексу сирої нафти

Деякі графіки будуть представляти парну побудову лише часового ряду та мультифрактального індикатора. Скористаємось функцією `plot_pair()`, що ми визначали в попередніх лабораторних:

```
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', rotation=45, **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();
```

7.2.0.1 Віконна процедура

Для подальших розрахунків знову скористаємось бібліотекою `fathon`, що ми використовували попередньо для виконання класичного аналізу детрендованих флуктуацій. Переваги саме цієї бібліотеки полягають у можливості використання процедури розрахунку розбиття ряду на сегменти починаючи з кінця ряду, оскільки довжина ряду не завжди дозволяє поділити його на локальні сегменти націло. Тобто, теоретично у нас залишається сегмент ряду, що не піддається розбиттю на локальні сегменти. Тому повторна процедура розбиття на локальні сегменти починаючи з кінця ряду дозволяє обійти дану проблему. Для спрощення представлення теоретичного матеріалу ми не стали імплементувати дану процедуру, але вона доступна в бібліотеці `fathon`. Окрім цього, бібліотека надає можливість обчислення крос-кореляційного аналізу детрендованих флуктуацій та його мультифрактального аналогу.

```
import fathon
from fathon import fathonUtils as fu
```

Приступимо до ініціалізації параметрів.

```
window = 500 # розмір вікна
tstep = 5    # 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

win_beg = 10       # Початкова ширина сегменту
win_end = window-1 # Кінцева ширина сегменту

scales_exp_wind = fu.linRangeByStep(win_beg, win_end) # генеруємо масив
# лінійно розділених
# елементів

rev = True # чи повторювати розрахунок ф-ції флуктуацій з кінця

length = len(time_ser.values)

q_min = -5  # мінімальне значення q
q_max = 5   # максимальне значення q
q_step = 1  # крок збільшення q

nq = np.arange(q_min,
               q_max+q_step,
               q_step)

order = 1 # порядок поліноміального тренду

delta_alph = []
delta_spec = []
max_alph = []
min_alph = []
mean_alph = []
alpha_zero = []
delta_alph_right = []
delta_alph_left = []
assym = []
delta_s = []
D_0 = []
D_1 = []
D_2 = []
D_left = []
D_right = []
C_q = []
h_q = []
tau_q = []
D_q = []
mfSpect = []
alpha = []
hFI = []
alphaCF = []
C_q_area_wind = []
```

Розпочнемо процедуру рухомого вікна, що об’єднуватиме у собі попередні етапи розрахунків:

```
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)

# знаходження кумулятивного ряду
    cumulative = fu.toAggregated(fragm)

# інціалізації процедури mfdfa
    pymfdfa = fathon.MFDFA(cumulative)

# обчислення функції флуктуацій та отримання узагальненого показника Херста
    n, F = pymfdfa.computeFlucVec(scales_exp_wind, nq, revSeg=rev, polOrd=order)
    Hq_fragm, _ = pymfdfa.fitFlucVec()

# отримання показника tau
    tau_wind = nq * Hq_fragm - 1

# отримання показника сингулярності
    alpha_wind = np.gradient(tau_wind, nq, edge_order=2)

# отримання мультифрактального спектра
    f_wind = nq * alpha_wind - tau_wind

# отримання мультифрактальної теплоємності
    C_q_wind = -np.gradient(alpha_wind, nq, edge_order=2)

# інтегральний показник C(q)
    C_q_area = cumulative_trapezoid(np.abs(C_q_wind), nq, initial=0)[-1]

# ширина спектра мультифрактальності
    delta_alpha_wind = alpha_wind.max() - alpha_wind.min()

# відстань між кінцями спектра мультифрактальності
    delta_phi = f_wind[-1] - f_wind[0]

# максимальне значення альфи
    maximal_alpha = alpha_wind.max()

# мінімальне значення альфи
    minimal_alpha = alpha_wind.min()

# середнє значення альфи
    mean_alpha = np.mean(alpha_wind)

# значення сингулярності при якому спектр приймає максимальне значення (α0)
    alpha_0 = alpha_wind[np.nanargmax(f_wind)]
```

```
# ширина правого хвоста спектра
    delt_alpha_right = maximal_alpha - alpha_0

# ширина лівого хвоста спектра
    delt_alpha_left = alpha_0 - minimal_alpha

# різниця між шириною лівого та правого хвостів
    delt_s = delt_alpha_right - delt_alpha_left

# показник асиметрії
    A = (delt_alpha_left - delt_alpha_right)/(delt_alpha_left +
delt_alpha_right)

# визначаємо індекс при q=0
    difference_zero = np.absolute(nq-0)
    idx_zero = difference_zero.argmin()

# визначаємо індекс при q=1
    difference_one = np.absolute(nq-1)
    idx_one = difference_one.argmin()

# визначаємо індекс при q=2
    difference_two = np.absolute(nq-2)
    idx_two = difference_two.argmin()

# ініціалізуємо масиви під розмірності
    Dq_wind = np.zeros(len(nq))

# Визначаємо узагальнені фрактальні розмірності там де q!=1
    Dq_wind[nq!=nq[idx_one]] = tau_wind[nq!=nq[idx_one]] / (nq[nq!=nq[idx_one]]-
1)

# Визначаємо окремо узагальнені фрактальні розмірності при q=1
    Dq_wind[nq==nq[idx_one]] = -tau_wind[nq==nq[idx_one]]

# Узагальнені фрактальні розмірності отримані з мультифрактального спектра
    D_zero = f_wind[nq==nq[idx_zero]]
    D_one = f_wind[nq==nq[idx_one]]
    D_two = 2*alpha_wind[nq==nq[idx_two]] -
f_wind[np.where(alpha_wind[nq==nq[idx_two]])]

# відстань від центру розподілу узагальнених розмірностей до лівого кінця
    delta_D_Q_left = Dq_wind[nq==q_min] - Dq_wind[nq==nq[idx_zero]]

# відстань від центру розподілу узагальнених розмірностей до правого кінця
    delta_D_Q_right = Dq_wind[nq==nq[idx_zero]] - Dq_wind[nq==q_max]

# індекс h-флуктуацій (hFI)
    fluct = np.sum(np.gradient(np.gradient(Hq_fragm, nq, edge_order=2), nq,
edge_order=2)**2)/(2*np.max(np.abs(nq))+2)

# кумулятивний індекс інкрементів узагальнених показників Херста (αCF)
    incr = np.sum(np.gradient(Hq_fragm, edge_order=2)**2/ np.gradient(nq,
edge_order=2))
```

```
    delta_alph.append(delta_alpha_wind)
    delta_spec.append(delta_phi)
    max_alph.append(maximal_alpha)
    min_alph.append(minimal_alpha)
    mean_alph.append(mean_alpha)
    alpha_zero.append(alpha_0)
    delta_alph_right.append(delt_alpha_right)
    delta_alph_left.append(delt_alpha_left)
    delta_s.append(delt_s)
    assym.append(A)
    D_0.append(D_zero)
    D_1.append(D_one)
    D_2.append(D_two)
    D_left.append(delta_D_Q_left)
    D_right.append(delta_D_Q_right)
    C_q.append(C_q_wind)
    mfSpect.append(f_wind)
    alpha.append(alpha_wind)
    hFI.append(fluct)
    alphaCF.append(incr)
    C_q_area_wind.append(C_q_area)
    h_q.append(Hq_fragm)
    tau_q.append(tau_wind)
    D_q.append(Dq_wind)
```

Зберігаємо абсолютні значення показників до текстових файлів.

```
# перелік назв кожного індикатора для збереження до txt
subtitle_of_txts = ['delta_alpha', 'delta_f', 'max_alpha', 'min_alpha',
'mean_alpha',
'zero_alpha', 'delta_alpha_right', 'delta_alpha_left', 'assymetry',
'delta_s', 'D_0', 'D_1', 'D_2', 'hFI', 'alphaCF', 'C_q_area',
'delta_d_left', 'delta_d_right']

# перелік вихідних значень індикаторів для збереження до txt
mfdfa_indicators = [delta_alph, delta_spec, max_alph, min_alph, mean_alph,
alpha_zero, delta_alph_right, delta_alph_left, assym, delta_s, D_0, D_1, D_2,
hFI, alphaCF, C_q_area_wind, D_left, D_right]

for i in range(len(subtitle_of_txts)):
    np.savetxt(f"mfdfa_{subtitle_of_txts[i]}_name={symbol}_ret={ret_type}_ \
                order={order}_qmin={q_min}_qmax={q_max}_qinc={q_step}_ \
              wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}.txt",
mfdfa_indicators[i])
```

Проаналізуємо динаміку отриманих індикаторів.

7.2.1 Ширина спектра мультифрактальності ($\Delta\alpha$)

Першим і одним із найпрактичніших індикаторів складності системи є **ширина спектра мультифрактальності** (multifractal width), $\Delta\alpha$, яку можна подати як різницю між максимальним ступенем сингулярності та мінімальним:

$$\Delta\alpha = \alpha_{max} - \alpha_{min}. \quad (7.10)$$

Якщо проводити аналогію з термодинамічними показниками, то в такому разі ширина спектра мультифрактальності становитиме різницю між найбільшим і найменшим значенням внутрішньої енергії системи. Розглянемо динаміку даного показника для ринку нафти:

```
measure_label = r'$\Delta\alpha$'
file_name =
f"mfdfa_delta_alpha_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax
={q_max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          delta_alph,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='red')
```

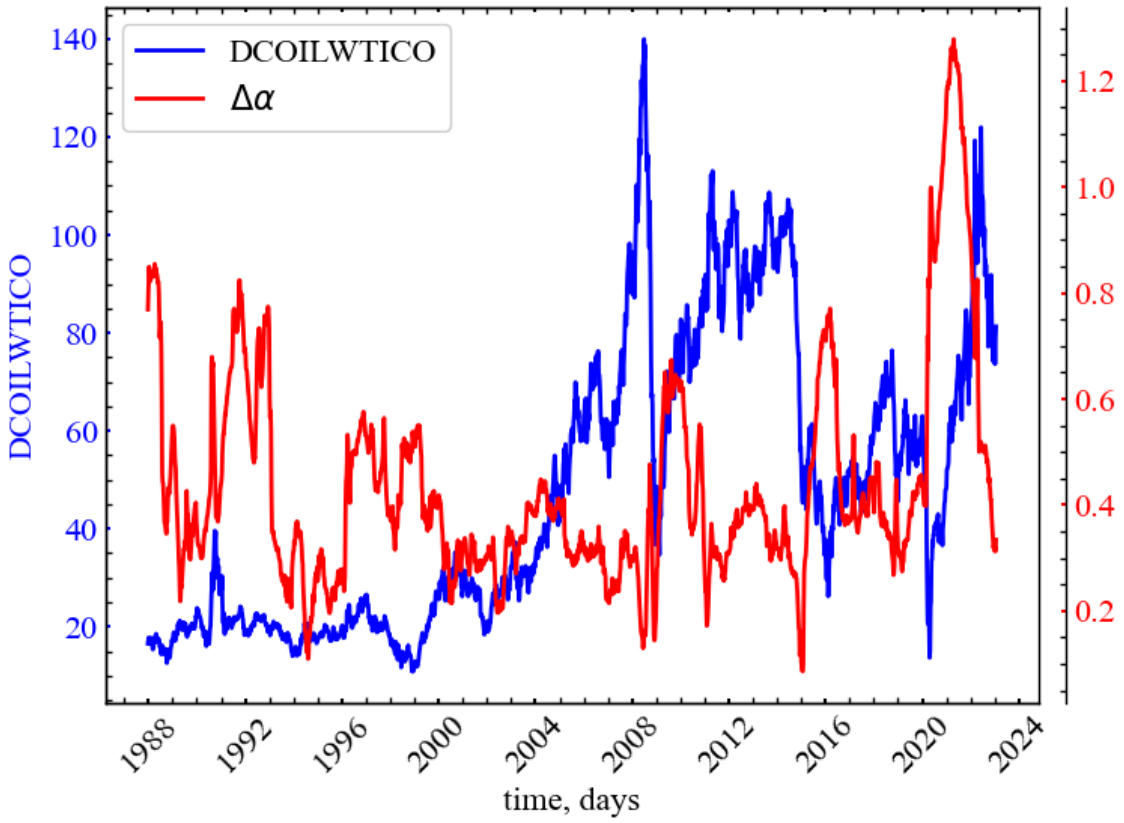


Рис. 7.25: Динаміка індексу сирої нафти WTI та показника ширини спектра мультифрактальності $\Delta\alpha$

На Рис. 7.25 видно, що ширина спектра мультифрактальності зростає під час кризових подій, що вказує на зростання загального ступеня складності та

періодизації. Тобто, даний показник слугує ще одним підтвердженням, що трейдери на ринку, наприклад, нафти поводять себе у синхронній манері в ході кризи. Зростання загального ступеня мультифрактальності є індикатором зростання кореляцій в системі, що підтверджувалось і попередніми індикаторами складності.

7.2.2 Різниця між кінцями спектра мультифрактальності (Δf)

Однак проста ширина спектра мультифрактальності не показує, наприклад, флуктуації якого типу є найвірогіднішими, елементи якого типу щільності відіграють найбільшу роль у зростанні або зниженні складності системи. Надалі було запропоновано такий показник мультифрактальності як Δf, який можна представити наступним чином [117–119]:

$$\Delta f = f(\alpha_{min}) - f(\alpha_{max}). \tag{7.11}$$

```
measure_label = r'$\Delta f$'
file_name =
f"mfdfa_delta_f_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax={q_
max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          delta_spec,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='brown')
```

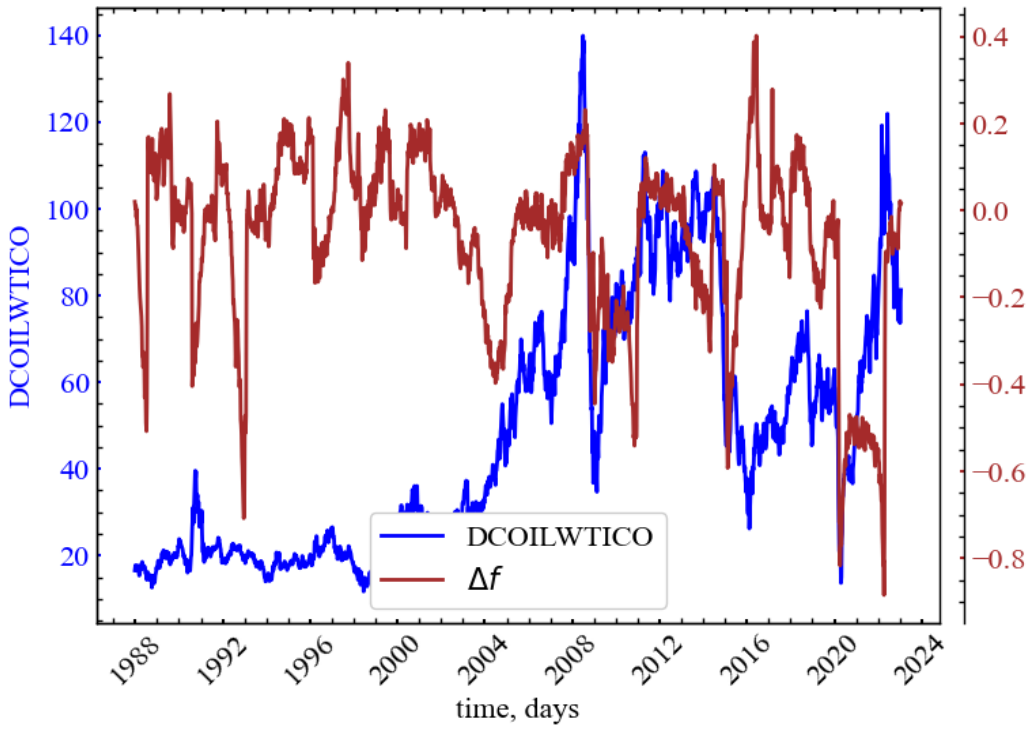


Рис. 7.26: Динаміка індексу сирої нафти WTI та показника відстані між кінцями спектра мультифрактальності Δf

Сенс даного показника полягає в тому, що він дає нам змогу визначити ступінь імовірності появи елементів з великими щільностями і малими. Якщо цей показник менший за нуль, тоді флуктуації, що відображають елементи з найбільшою концентрацією (найбільшими фруктуаціями), мають найбільшу ймовірність. Якщо цей показник вищий за нуль, тоді флуктуації, що відображають малоконцентровані елементи (малі флуктуації), визначають динаміку системи. Якщо цей показник перебуває в нулі, тоді як високосингулярні, так і малосингулярні елементи мають рівномірний внесок у динаміку системи.

Звертаючись до термодинаміки, можна згадати, що $f(\alpha)$ — це ентропія системи. Тоді стає зрозуміло, що варіативність спектра мультифрактальності дає нам змогу визначити ступінь внеску висококонцентрованих і малоконцентрованих елементів у мінімізацію ентропії системи. Лівостороння асиметрія мультифрактального спектра $(\Delta f > 0)$ підказує нам, що висококонцентровані елементи фазового простору дають найбільший внесок у мінімум термодинамічної ентропії. Іншими словами, ці елементи і є двигуном зростання впорядкованості динаміки системи. Своєю чергою, правостороння асиметрія мультифрактального спектра $(\Delta f < 0)$ вказує на мінімізацію ентропії за рахунок малоконцентрованих елементів. Симетрія кінців спектра вказує на рівний внесок високощільних і розріджених областей на мінімізацію ентропії.

Як уже згадувалося, трапляються випадки, коли мультифрактальний спектр практично сходиться в сингулярність (одну точку). У такому разі ми маємо справу з простою монофрактальною системою, яка в нашому випадку характеризувалася незалежними і нормально розподіленими випадковими величинами. Для такого спектра і $\Delta\alpha$, і Δf будуть прямувати до нуля. Для такого часового ряду вже спостерігається не множина фрактальних розмірностей, а тільки один фрактальний показник, $f[\alpha(q=0)]=1$. Це саме та область, де система досягає своєї термодинамічної рівноваги — максимуму ентропії. У свою чергу Δf можна охарактеризувати як різницю ентропій за граничної максимальної та мінімальної внутрішньої енергії системи.

7.2.3 Ширина лівого ($\Delta\alpha_L$) та правого ($\Delta\alpha_R$) хвостів мультифрактального спектра

Крім цього, ми можемо дослідити ступінь складності динаміки окремо високощільних областей (з великими флуктуаціями) і низькощільних (з малими флуктуаціями). Для цього виміряємо **ширину окремо лівого** і **правого хвостів**. Ширину лівого хвоста визначимо як

$$\Delta\alpha_L = \alpha_0 - \alpha_{min}, \tag{7.12}$$

а ширину правого хвоста як

$$\Delta\alpha_R = \alpha_{max} - \alpha_0. \tag{7.13}$$

У свою чергу ширина лівого хвоста вимірює ступінь складності флуктуацій з великою амплітудою, а ширина правого хвоста — малих. Зростання ширини кожного з хвостів відображатиме зростання ступеня кореляцій між елементами.

```
fig, ax = plt.subplots(1, 1)

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.index[window:length:tstep],
              time_ser.values[window:length:tstep],
"b-", label=fr"{ylabel}")
p2, = ax2.plot(time_ser.index[window:length:tstep],
               delta_alph_left, color="r", label=r"$\Delta\alpha_{L}$")
p3, = ax3.plot(time_ser.index[window:length:tstep],
               delta_alph_right, color="g", label=r"$\Delta\alpha_{R}$")
```

```
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())

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)
ax.tick_params(axis='x', rotation=45, **tkw)

ax3.legend(handles=[p1, p2, p3])

plt.savefig(f"mfdfa_delta_alpha_left_right_name={symbol}_ret={ret_type}_order={o
rder}_qmin={q_min}_qmax={q_max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}.jpg")
plt.show();
```

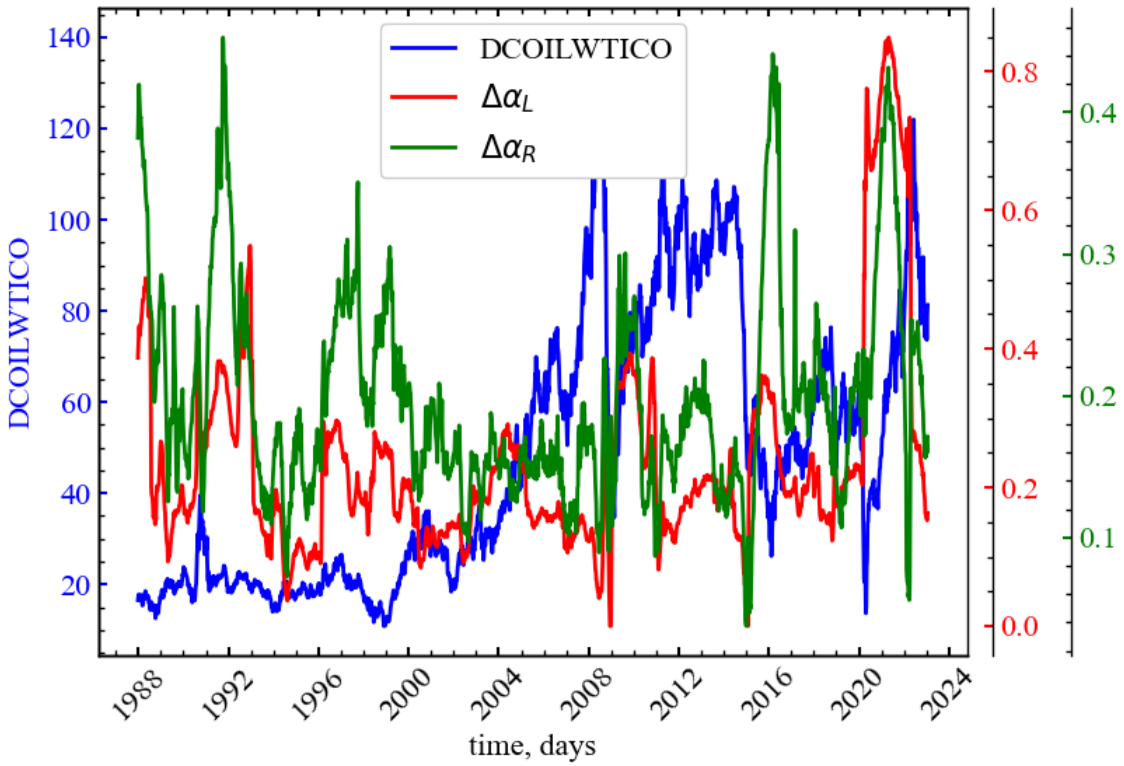


Рис. 7.27: Динаміка індексу сирої нафти WTI та ширини лівого і правого хвостів мультифрактального спектра

Із Рис. 7.27 видно, що досліджувані індикатори реагують у характерний спосіб на кризові події. Ширина лівої сторони спектра мультифрактальності зростає під час 1992, 1996-2000, 2008, 2016 років та коронавірусної пандемії. Це вказує на домінацію висококонцентрованих флуктуацій (з великою амплітудою коливань). Окрім цього, зростання ширини лівого хвоста вказує на те, що

флуктуації з великою амплітудою коливань характеризуються зростанням ступеня кореляцій під час кризових подій, що в свою чергу може слугувати індикатором зростання процесів самоорганізації.

Хоча і меншою, але не менш примітною є динаміка ширини правого хвоста мультифрактального спектра. Цей індикатор працює майже аналогічно до ширини лівого хвоста, але характеризує динаміку малоконцентрованих величин — флуктуацій з малою амплітудою коливань. Майже синхронна динаміка обох показників указує на зростання впливу коливань як флуктуацій з великою амплітудою коливань, так і флуктуацій з малою амплітудою коливань. Тобто, два типи флуктуацій є джерелом зростання нелінійних кореляцій під час кризових подій.

7.2.4 Показник сингулярності α та його різновиди

В якості можливих індикаторів складності системи можна взяти α_{min}, α_{max}, α_{mean} і α_0, які, відповідно, характеризують **мінімальну силу сингулярності**, **максимальну**, **середню** і **сингулярність за умови рівноважного врахування** як великих флуктуацій, так і малих.

```
fig, ax = plt.subplots(1, 1)

ax2 = ax.twinx()
ax3 = ax.twinx()
ax4 = ax.twinx()
ax5 = ax.twinx()

ax3.spines.right.set_position(("axes", 1.08))
ax4.spines.right.set_position(("axes", 1.18))
ax5.spines.right.set_position(("axes", 1.27))

p1, = ax.plot(time_ser.index[window:length:tstep],
time_ser[window:length:tstep], "b-", label=fr"{ylabel}")
p2, = ax2.plot(time_ser.index[window:length:tstep], max_alph, "r-",
label=r"$\alpha_{max}$")
p3, = ax3.plot(time_ser.index[window:length:tstep], min_alph, "g-",
label=r"$\alpha_{min}$")
p4, = ax4.plot(time_ser.index[window:length:tstep], mean_alph, "c-",
label=r"$\alpha_{mean}$")
p5, = ax5.plot(time_ser.index[window:length:tstep], alpha_zero, "m-",
label=r"$\alpha_{0}$")

ax.set_xlabel(xlabel)
ax.set_ylabel(fr"{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())
```

```
ax5.yaxis.label.set_color(p5.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)
ax5.tick_params(axis='y', colors=p5.get_color(), **tkw)
ax.tick_params(axis='x', **tkw, pad=10, rotation=45)

ax5.legend(handles=[p1, p2, p3, p4, p5])

plt.savefig(f"mfdfa_alpha_min_max_mean_zero_name={symbol}_ret={ret_type}_order={
order}_qmin={q_min}_qmax={q_max}_qinc={q_step}_ \
            wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}.jpg",
bbox_inches="tight")
```

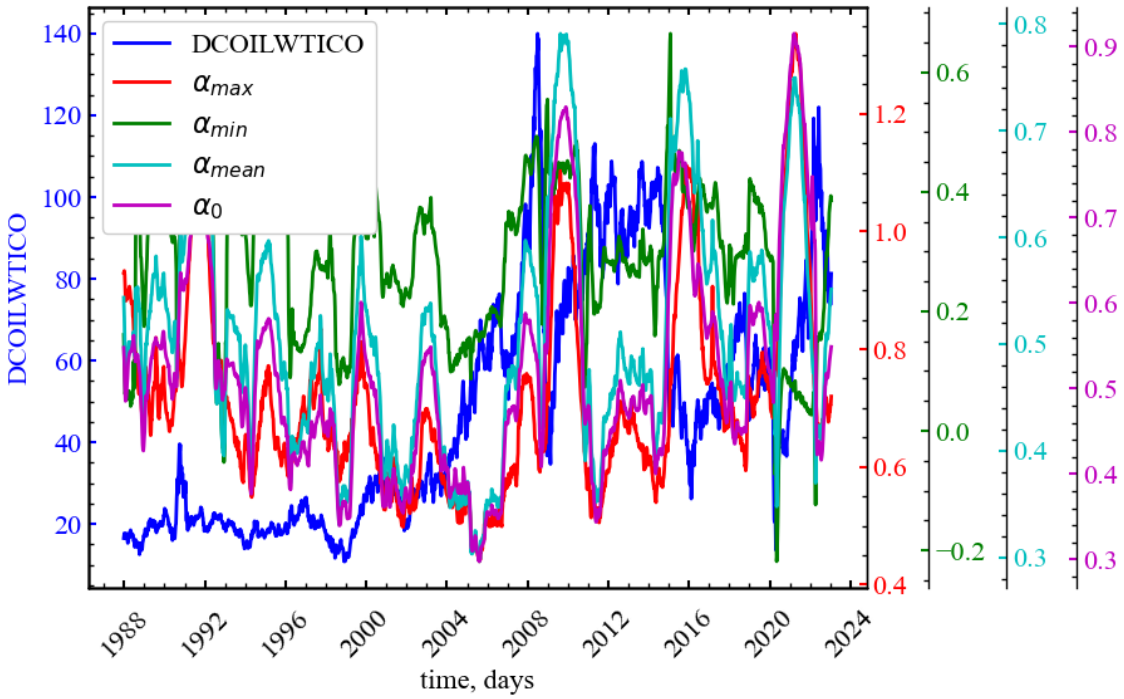


Рис. 7.28: Динаміка індексу сирої нафти WTI та показників сингулярності

Як видно з Рис. 7.28, усі показники сингулярності зростають в області фінансового фазового переходу зі стану стабільності до стану кризи. Це говорить про зростання складності системи: різкому приросту кількості агентів, що задіяні в самоорганізованому розвитку досліджуваної системи. З погляду термодинаміки можна було б сказати, що при фінансових крахових подіях зростає внутрішня енергія системи.

7.2.5 Тип довгого хвоста мультифрактального спектра (ΔS)

Крім такої міри як, наприклад, Δf можна представити й інші міри асиметрії мультифрактального спектра. Наприклад, ми можемо визначити **тип довгого хвоста мультифрактального спектра** за допомогою показника ΔS [120]:

$$\Delta S = \Delta\alpha_R - \Delta\alpha_L. \tag{7.14}$$

Якщо $\Delta S < 0$, мультифрактальний спектр має довгий лівий хвіст, що свідчить про чутливість часового ряду до локальних флуктуацій з великою амплітудою. Якщо $\Delta S > 0$, мультифрактальний спектр має довгий правий хвіст, який вказує на чутливість структури сигналу до локальних флуктуацій з малою амплітудою коливань. У тих випадках, коли високо- і низькофлуктуаційні компоненти сигналу співставні, спектр сингулярностей буде приблизно симетричним і $\Delta\alpha_R = \Delta\alpha_L$.

```
measure_label = r'$\Delta S$'
file_name =
f"mfdfa_delta_s_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax={q_
max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          delta_s,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkorange')
```

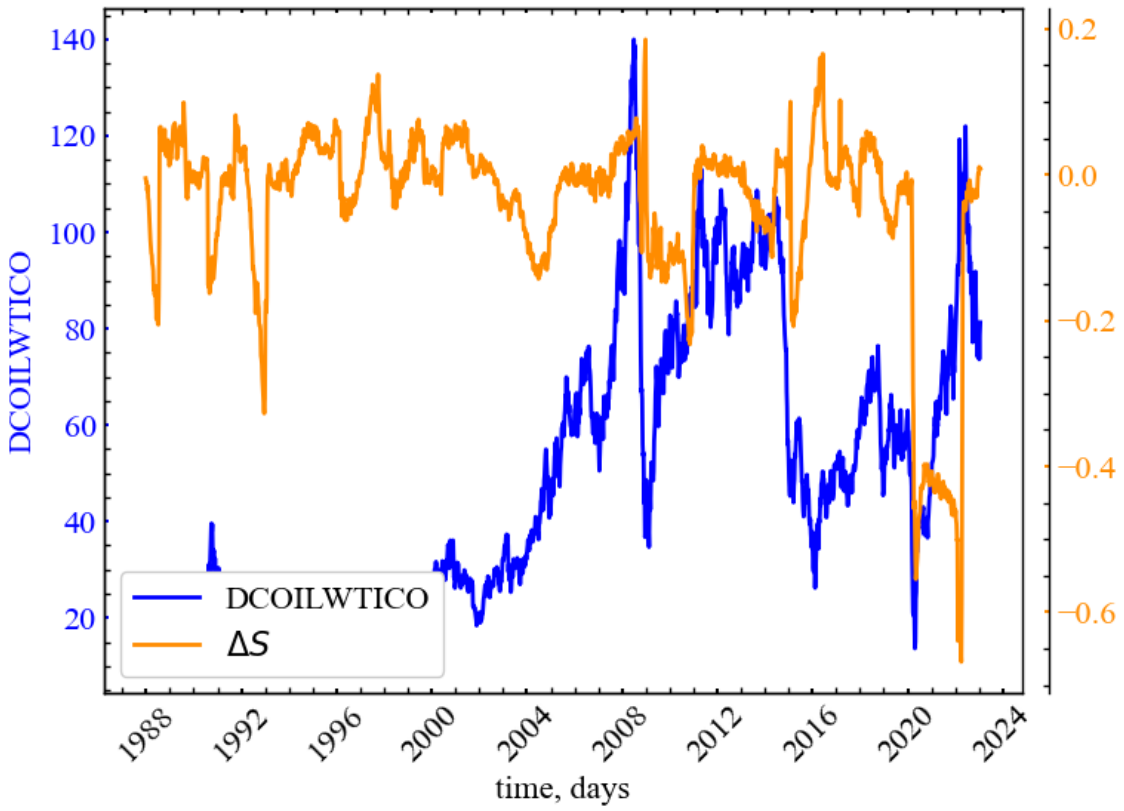


Рис. 7.29: Динаміка індексу сирої нафти WTI та показника типу хвоста мультифрактального спектра ΔS

З огляду на Рис. 7.29 видно, що $\Delta S < 0$, що свідчить про те, що найбільш крахові ділянки нафтового ринку зумовлені флуктуаціями з великою амплітудою коливань. Особливо помітними тут предстають кризи 1992 та 2020 років.

7.2.6 Показник асиметрії A

Далі ми можемо визначити наступний параметр асиметрії [121–123]:

$$A = (\Delta\alpha_L - \Delta\alpha_R)/(\Delta\alpha_R + \Delta\alpha_L) = -\Delta S/\Delta\alpha. \qquad (7.15)$$

Параметр асиметрії пов'язаний з предомінуючим типом коливань у досліджуваній системі. Якщо $A = 0$ ($\Delta\alpha_L = \Delta\alpha_R$), динаміку системи представляє симетричний спектр. Якщо $A < 0$ ($\Delta\alpha_L < \Delta\alpha_R$), мультифрактальний спектр має правосторонню асиметрію, що підкреслює сильніший вплив малих флуктуацій на мультифрактальність. І навпаки, коли $A > 0$ ($\Delta\alpha_L > \Delta\alpha_R$), тоді ми маємо справу з лівостороннім спектром, який позначає більшу неоднорідність для великих флуктуацій і вказує на те, що в часовому ряді переважає мультифрактальна природа неоднорідностей з високими щільностями. Оскільки

асиметрію виявляють за знаком A, що еквівалентний знаку ΔS, то, ґрунтуючись тільки по знаку ΔS, можна зробити висновки як про тип довгого хвоста, так і про знак показника A мультифрактального спектра, тобто нечутливість і тип домінантних коливань мультифрактальності часового ряду.

```
measure_label = r'$A$'
file_name =
f"mfdfa_A_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax={q_max}_q
inc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          assym,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkviolet')
```

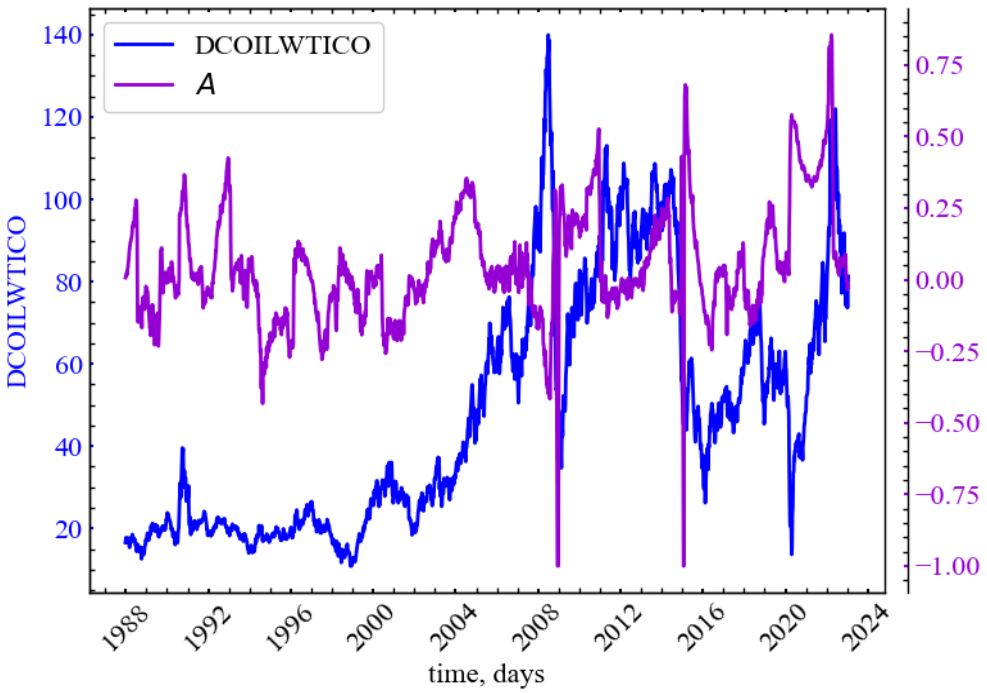


Рис. 7.30: Динаміка індексу сирої нафти WTI та показника асиметрії мультифрактального спектра A

З Рис. 7.30 видно, що, як правило, показник асиметрії зростає під час крахових подій і вказує на домінацію лівостороннього спектра (висококонцентрованих флуктуацій з великою амплітудою коливань). Окрім цього видно, що, наприклад, для криз 2008 та 2015-2016 років спостерігалась помітна короткочасна правостороння асиметрія, що характеризується швидкоплинним сплеском показника асиметрії у сторону від'ємних значень. Асоціювати малі та великі флуктуації з конкретними настроями або

поведінковими патернами ринку доволі складно. На даний момент ми можемо відзначити тільки те, що дані події представляли найбільш багату варіацію як короткострокових, так і довгострокових кореляцій.

7.2.7 Індекс h-флуктуацій (hFI)

Флуктуацію можна проаналізувати за допомогою другої похідної узагальненого показника Херста. Зауважимо, що амплітуда другої похідної у випадку мультифрактальних сигналів більша, ніж для монофрактальних. Для одержання потрібної інформації з $h(q)$ було запропоновано **h-індекс флуктуації** (hFI) [124], що визначається як степінь другої похідної від $h(q)$:

$$hFI = \frac{1}{2|q_{max}| + 2} \sum_{q=q_{min}-2}^{q_{max}} [h(q) - 2h(q-1) + h(q-2)]^2. \qquad (7.16)$$

Чим вище значення даного показника, тим вища самоорганізованість системи.

```
measure_label = r'$hFI$'
file_name =
f"mfdfa_hFI_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax={q_max}
_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          hFI,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='green')
```

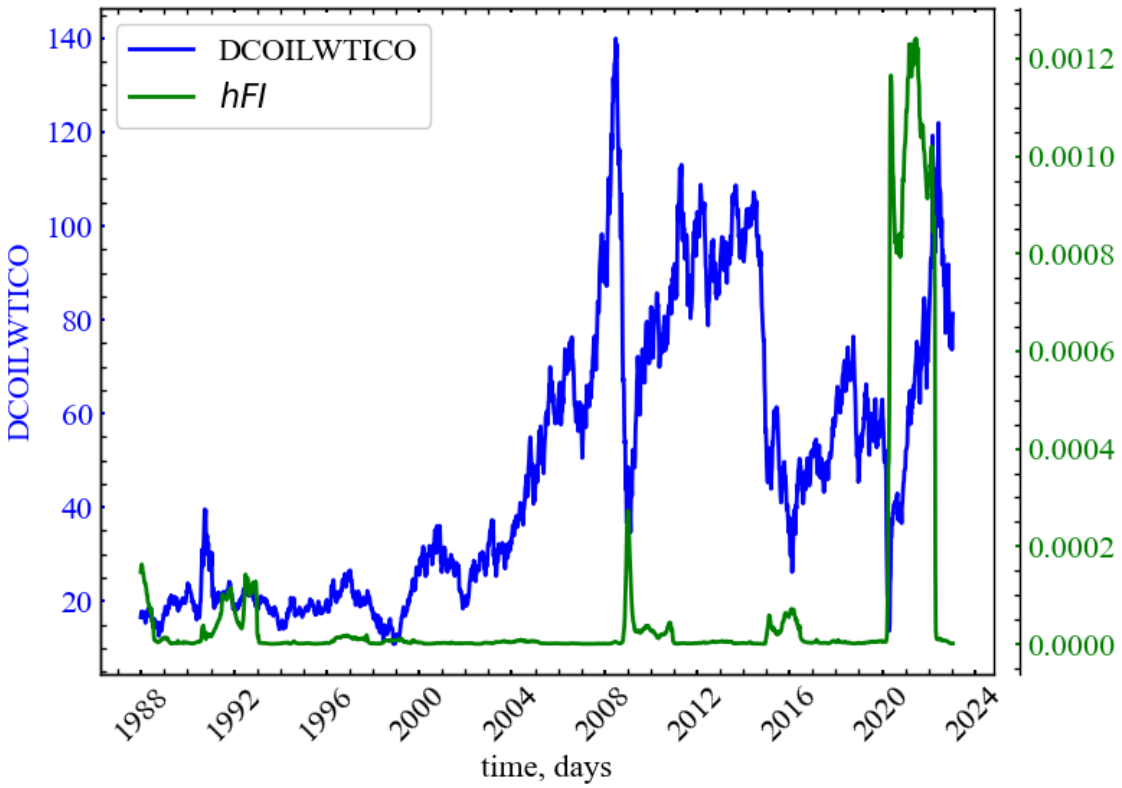


Рис. 7.31: Динаміка індексу сирої нафти WTI та індексу h-флуктуацій hFI

Видно, що за hFI найвищий ступінь мультифрактальності проявляється саме для криз 1992, 2008, 2016 та 2020 років. Дані крахові події включають в себе найбільшу кількість різноманітних факторів, що впливали на динаміку досліджуваної системи. Особливо помітно це для коронавірусної пандемії.

7.2.8 Кумулятивний індекс інкрементів узагальнених показників Херста (αCF)

Кумулятивна функція квадратів інкрементів (αCF) [125] узагальнених показників Херста між послідовними моментними порядками є більш надійним показником розподілу узагальнених показників Херста.

```
measure_label = r'$\alpha CF$'
file_name =
f"mfdfa_alphaCF_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax={q_
max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          alphaCF,
          ylabel,
          measure_label,
          xlabel,
```

```
            file_name,
            clr='crimson')
```

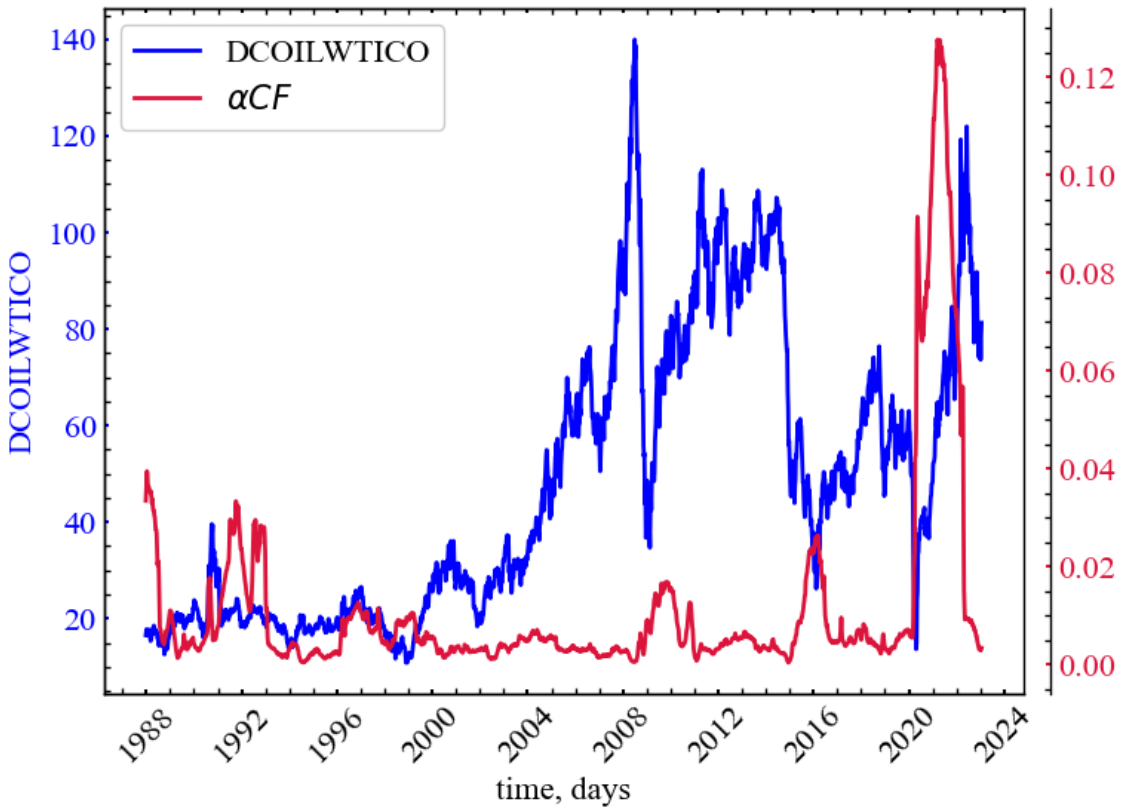


Рис. 7.32: Динаміка індексу сирої нафти WTI та кумулятивний індексу інкрементів узагальнених показників Херста αCF

Представлений кумулятивний індекс дещо відрізняється від hFI, але за логікою приблизно однаковий: події з найвищим ступенем мультифрактальності характеризуються вищою амплітудою αCF. Представлений показник виділяє ті самі кризи, що й попередній, але динаміка цього показника більш виразна, що робить його більш надійним для ідентифікації періодів самоорганізації системи.

7.2.9 Інтегральна мультифрактальна теплоємність $C(q)$

Загальний ступінь мультифрактальності, **інтегральну мультифрактальну питому теплоємність** (C_{area}), можна виразити через рівняння (7.17):

$$C_{area} = \int C(q)dq. \tag{7.17}$$

Розрахуємо і проаналізуємо динаміку цього показника.

```
measure_label = r'$C_{area}$'
file_name =
f"mfdfa_C_q_area_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax={q
```

```
_max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          C_q_area_wind,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkslateblue')
```

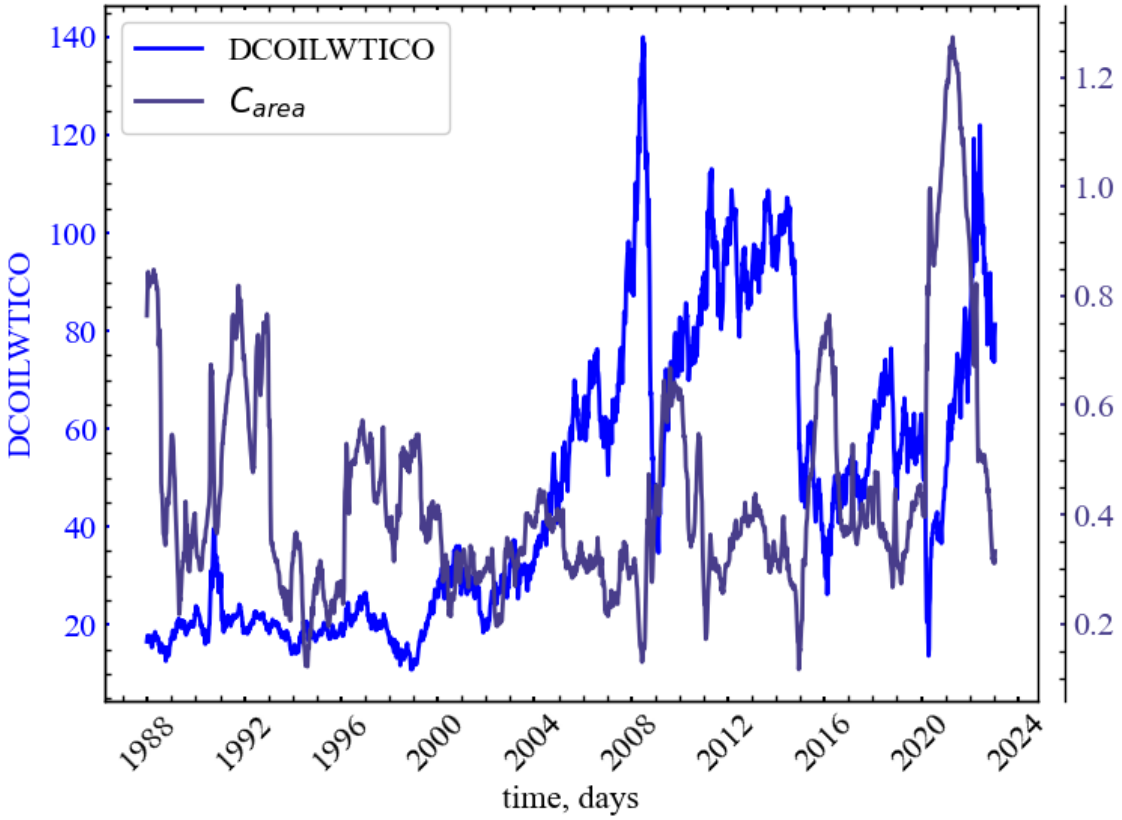


Рис. 7.33: Динаміка індексу сирої нафти WTI та інтегральної мультифрактальної теплоємності C_{area}

З представленого рисунку видно, що динаміка інтегральної теплоємності дуже подібна до ширини спектра мультифрактальності. Тобто, C_{area} є показником складності, що вказує на ступінь самоорганізованості фінансового фазового переходу. Видно, що фінансові крахи представляють доволі трендостійку динаміку, що є наслідком цілеспрямованих та колективних дій трейдерів на ринку.

7.2.10 Хаусдорфова розмірність (D_0)

Як уже зазначалось, D_0 представляє верхню межу змін розмірностей фрактальних підмножин атрактора системи. Інформацію про статистичні властивості системи він не містить і не представляє особливої цінності.

```
measure_label = r'$D_{0}$'
file_name =
f"mfdfa_D_0_area_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax={q
_max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          D_0,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkred')
```

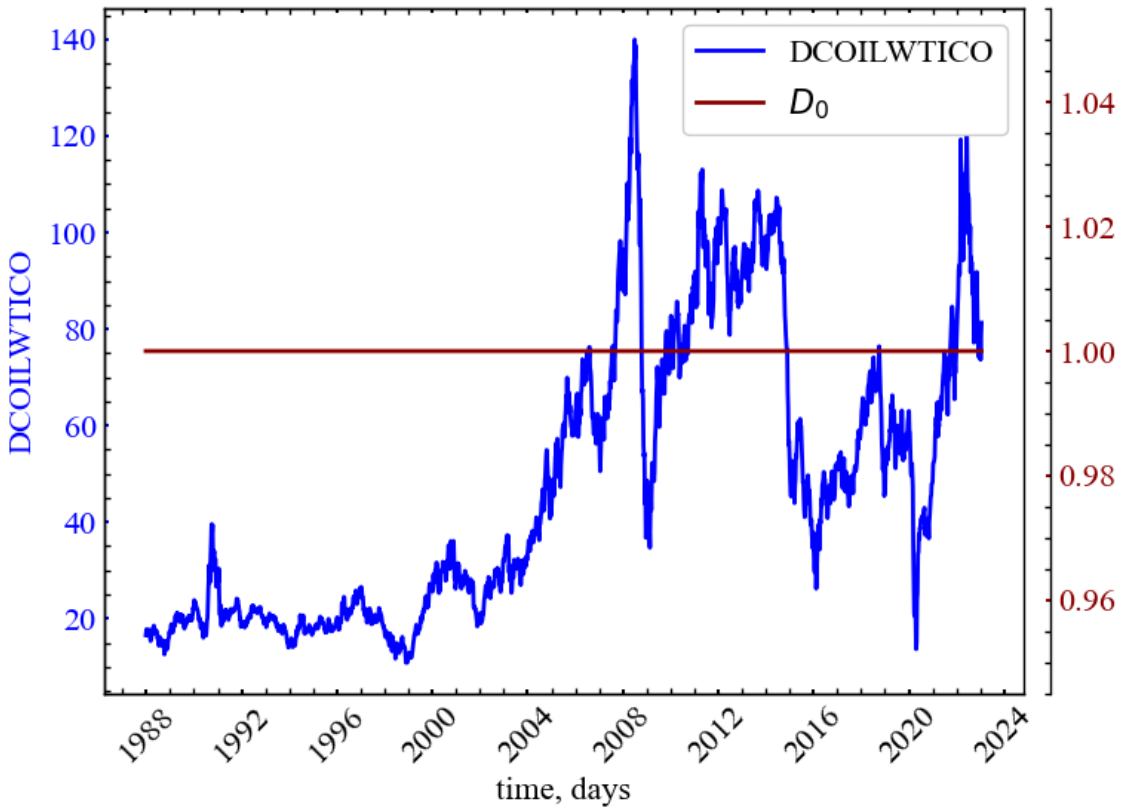


Рис. 7.34: Динаміка індексу сирої нафти WTI та хаусдорфової розмірності цього часового сигналу D_0

7.2.11 Інформаційна розмірність (D_1)

Інформаційна розмірність тісно пов'язана з інформаційною ентропією Шеннона. Чим вище значення D_1, тим швидше збільшується ентропія, що є показником того, наскільки мало ми знаємо про поточний стан системи. Зі зменшенням D_1 ентропія знижується, що, своєю чергою, свідчить про збільшення асиметрії в просторі, зменшення складності та розширення нашого розуміння поточного стану системи. Це також означає, що для опису можливих конфігурацій системи нам буде потрібно менше інформації.

```
measure_label = r'$D_{1}$'
file_name =
f"mfdfa_D_1_area_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax={q
_max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          D_1,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkred')
```

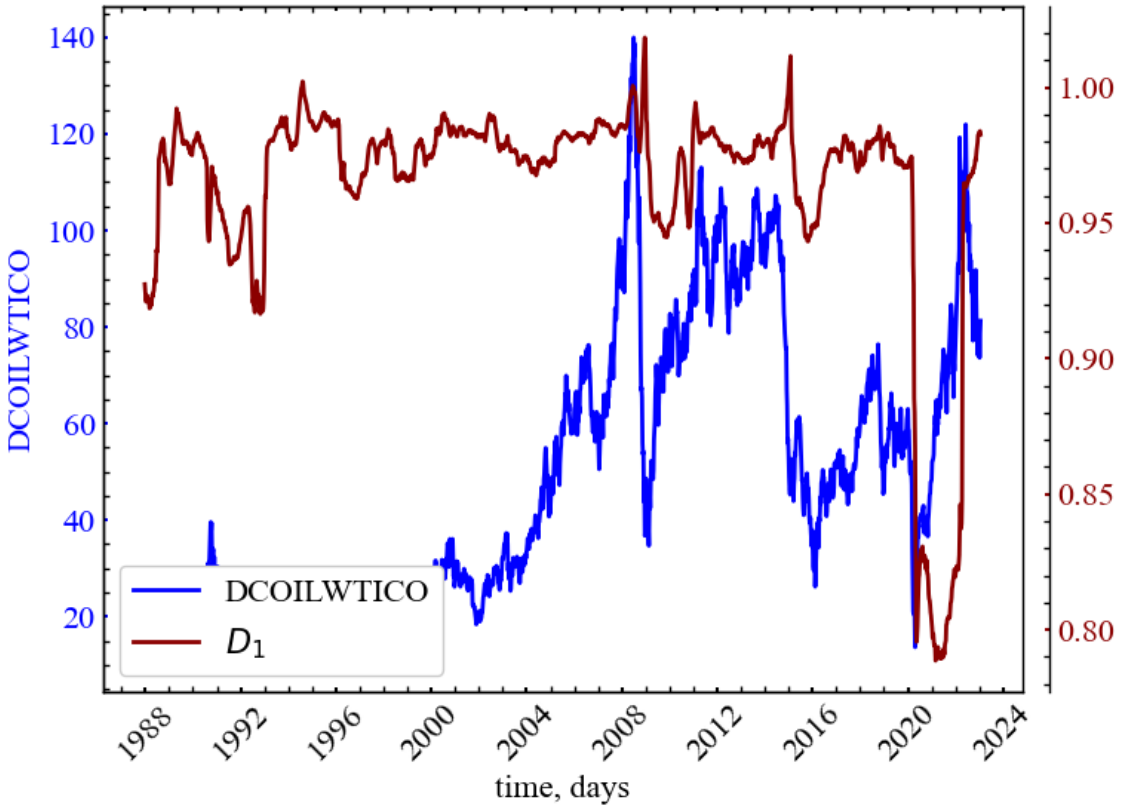


Рис. 7.35: Динаміка індексу сирої нафти WTI та інформаційної розмірності цього

часового сигналу D_1

На Рис. 7.35 видно, що інформаційна розмірність характеризується спадом під час крахових подій. Це вказує на зростання ступеня впорядкованості системи та колективної атракції агентів ринку до конкретної області фазового простору досліджуваної системи.

7.2.12 Кореляційна розмірність (D_2)

Кореляційну розмірність, аналогічно інформаційній розмірності, можна подати як тангенс кута нахилу лінії регресії, побудованої в логарифмічному масштабі, щодо залежності кореляційного інтеграла $C(\varepsilon)$ від ε. Подібно до D_1, кореляційна розмірність також визначає, як швидко змінюється значення кореляційного інтеграла.

```
measure_label = r'$D_{2}$'
file_name =
f"mfdfa_D_2_area_name={symbol}_ret={ret_type}_order={order}_qmin={q_min}_qmax={q
_max}_qinc={q_step}_\
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}"

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          D_2,
          ylabel,
          measure_label,
          xlabel,
          file_name,
          clr='darkred')
```

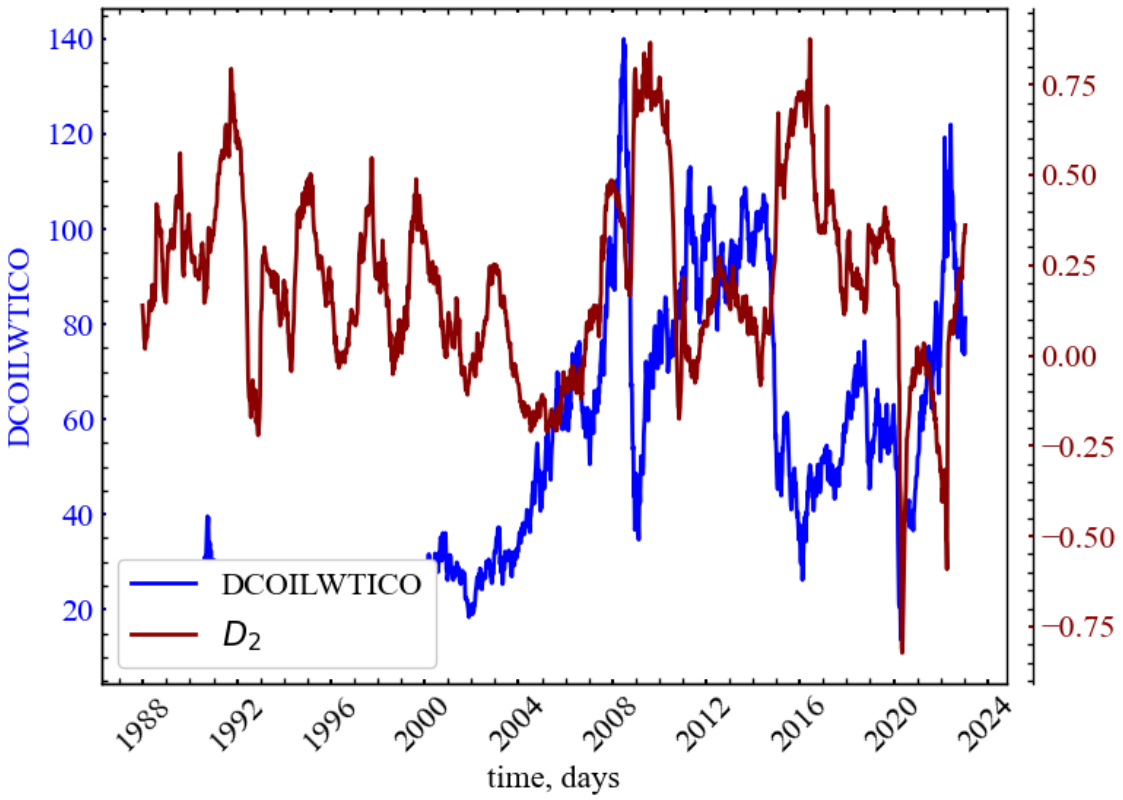


Рис. 7.36: Динаміка індексу сирої нафти WTI та кореляційної розмірності цього часового сигналу D_2

Кореляційна розмірність на Рис. 7.36 характеризується зростанням у передкризовий період та спадом під час кризи. Це говорить про те, що більшість агентів ринку починає орієнтуватися на один конкретний вектор розвитку системи.

7.2.13 Кривизна лівого (ΔD_L) та правого (ΔD_R) хвостів розподілу узагальнених фрактальних розмірностей

Охарактеризувати ступінь цієї складності можна за **кривизною окремо правого** та **лівого хвостів** узагальнених фрактальних розмірностей. Праву сторону (ΔD_R) можна визначити як

$$\Delta D_R = D_0 - D_{q_{max}}. \tag{7.18}$$

І чим більшим буде значення цієї міри, тим сильнішим буде ступінь впливу елементів із найбільшою концентрацією (щільністю, амплітудою флуктуацій) на загальну складність системи.

Кривизна лівого хвоста кривої узагальнених фрактальних розмірностей (ΔD_L):

$$\Delta D_L = D_{q_{min}} - D_0. \quad (7.19)$$

Цей показник буде говорити нам про те, наскільки сильним є вплив найменш концентрованих елементів на складність системи.

```
fig, ax = plt.subplots(1, 1)

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.index[window:length:tstep],
              time_ser.values[window:length:tstep],
"b-", label=fr"{ylabel}")
p2, = ax2.plot(time_ser.index[window:length:tstep],
               D_left, color="g", label=r"$\Delta D_{L}$")
p3, = ax3.plot(time_ser.index[window:length:tstep],
               D_right, color="r", label=r"$\Delta D_{R}$")


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())

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)
ax.tick_params(axis='x', rotation=45, **tkw)

ax3.legend(handles=[p1, p2, p3])

plt.savefig(f"mfdfa_delta_D_left_right_name={symbol}_ret={ret_type}_order={order
}_qmin={q_min}_qmax={q_max}_qinc={q_step}_ \
           wind={window}_step={tstep}_windbeg={win_beg}_winden={win_end}.jpg")
plt.show();
```

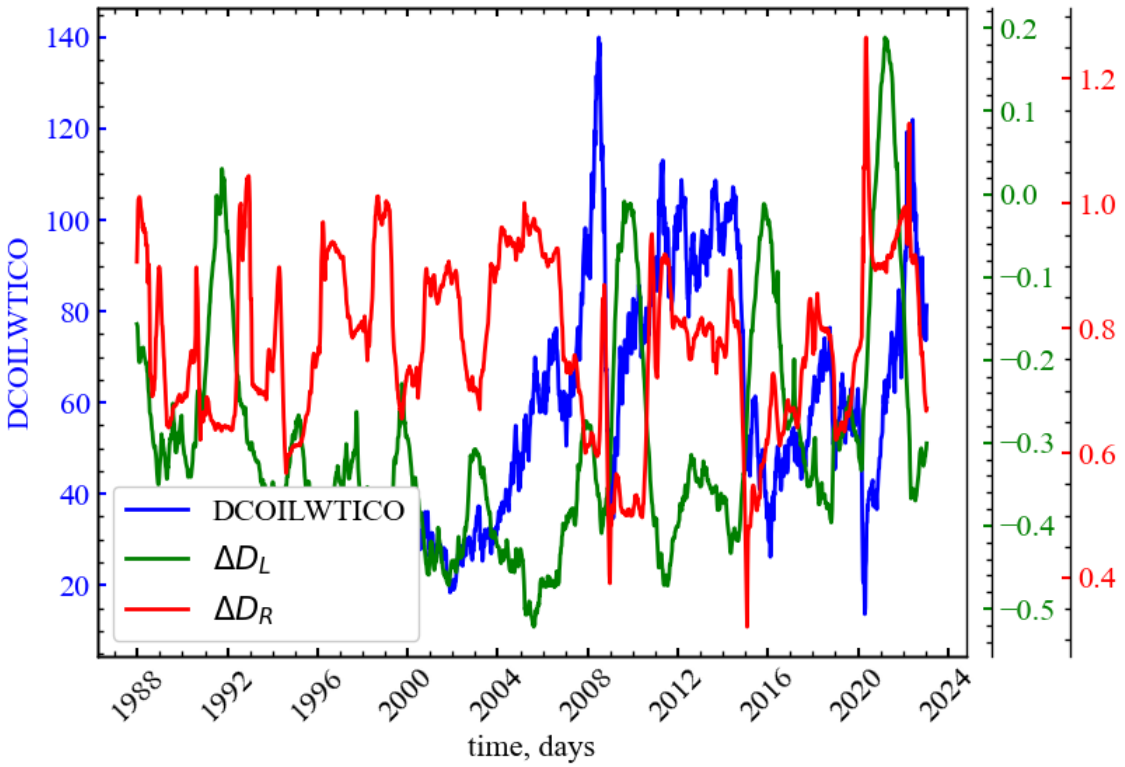


Рис. 7.37: Індекс сирої нафти WTI та кривизна лівого і правого хвостів спектра узагальнених фрактальних розмірностей

На рисунку (Рис. 7.37) видно, що спостерігаються етапи, при який два показники можуть вести себе як асинхронно, так і синхронно. Для 1992 року видно, що у передкризовий етап спостерігалась короткочасна домінація великих флуктуацій, на що і вказує ΔD_R. Під час краху спостерігалась домінація малоконцентрованих елементів, що демонструє зростання ΔD_L. Для 2001 року бачимо зростання впливу правого хвоста і зменшення впливу лівого. Для краху 2008 року видно зростання впливу висококонцентрованих елементів напередодні кризи. Після цього домінують малоконцентровані елементи. Бачимо зростання впливу малоконцентрованих елементів під час кризи 2016 року, і закономірне зменшення участі великоконцентрованих елементів. Пандемія 2020-2021 років характеризувалась активною участю як малоконцентрованих елементів, так і великоконцентрованих, на що вказує зростання обох показників кривизни хвостів узагальнених розмірностей.

7.2.14 Дво- та тривимірна візуалізація показників мультифрактальності

Попередньо ми аналізували залежності $h(q)$, $\tau(q)$, $D(q)$, $C(q)$ та $f(\alpha)$ для всього часового ряду. Тепер, скориставшись процедурою ковзного вікна, ми

можемо подивитися на їх зміну з плином часу. Скористаємось часовим рядом цін на нафту.

Перш за все оголосимо функції для побудови двовимірних графіків:

```
def plot_2d(X, Y, Z, subtitle_jpg, subtitle_fig, ylabel, barlabel, cmap, lims):

    fig, ax = plt.subplots(1, 1, figsize=(10, 5))

    cp = ax.contourf(X, Y, Z, alpha=0.8, cmap=cmap)
    plt.colorbar(cp, ax=ax, extend='both', label=barlabel)

    ax.set_xlim((time_ser.index[window:length:tstep][0],
                 time_ser.index[window:length:tstep][-1]))
    ax.set_ylim((np.min(lims), np.max(lims)))

    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)

    ax.set_title(subtitle_fig, pad=10)

    ax.tick_params(axis='both', which='major', pad=10)

    fig.tight_layout()


plt.savefig(f"mfdfa_{subtitle_jpg}_name={symbol}_ret={ret_type}_order={order}_ \

qmin={q_min}_qmax={q_max}_qinc={q_step}_windbeg={win_beg}_winden={win_end}.jpg",
                bbox_inches="tight")
    plt.show();
```

та тривимірних:

```
def plot_3d(X, Y, Z, subtitle_jpg, ylabel, zlabel, cmap):

    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

    surf = ax.plot_surface(X, Y, Z, cmap=cmap, rstride=2, cstride=2,
linewidth=0)

    ax.set_xlabel(xlabel, labelpad=15)
    ax.set_ylabel(ylabel, labelpad=15)
    ax.set_zlabel(zlabel, labelpad=15)
    ax.tick_params(axis='both', which='major', pad=5)

    fig.colorbar(surf, shrink=0.5, aspect=10, location='right', pad=0.1)

    fig.tight_layout()


plt.savefig(f"mfdfa_{subtitle_jpg}_name={symbol}_ret={ret_type}_order={order}_ \
                qmin={q_min}_qmax={q_max}_qinc={q_step}_windbeg={win_beg}_ \
                winden={win_end}.jpg", bbox_inches="tight")
```

```
    plt.show();
```

Після оголошення необхідний функцій можна приступати до візуалізації.

7.2.14.1 Динаміка $h(q)$ з ходом часу в дво- та тривимірному просторах

```
X, Y = np.meshgrid(time_ser.index[window:length:tstep], nq)
Z = np.array(h_q).T

plot_2d(X, Y, Z,
        subtitle_jpg='contour_h(q)',
        subtitle_fig=fr"Теплова діаграма $h(q)$",
        ylabel=r"$q$",
        barlabel=r"$h(q)$",
        cmap='jet',
        lims=nq)
```

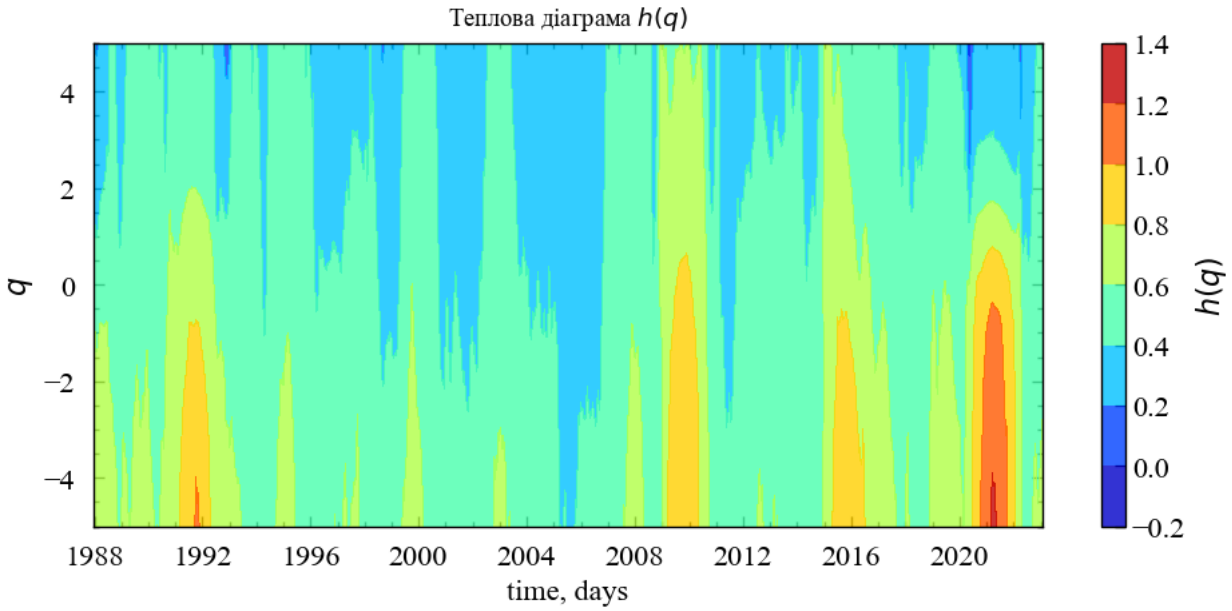


Рис. 7.38: Двовимірна контурна діаграма динаміки узагальненого показника Херста $h(q)$, що змінюється з плином часу

```
X, Y = np.meshgrid(np.arange(window, length, tstep), nq)
Z = np.array(h_q).T

plot_3d(X, Y, Z,
        subtitle_jpg='3d_h(q)',
        ylabel=r"$q$",
        zlabel=r"$h(q)$",
        cmap='jet')
```

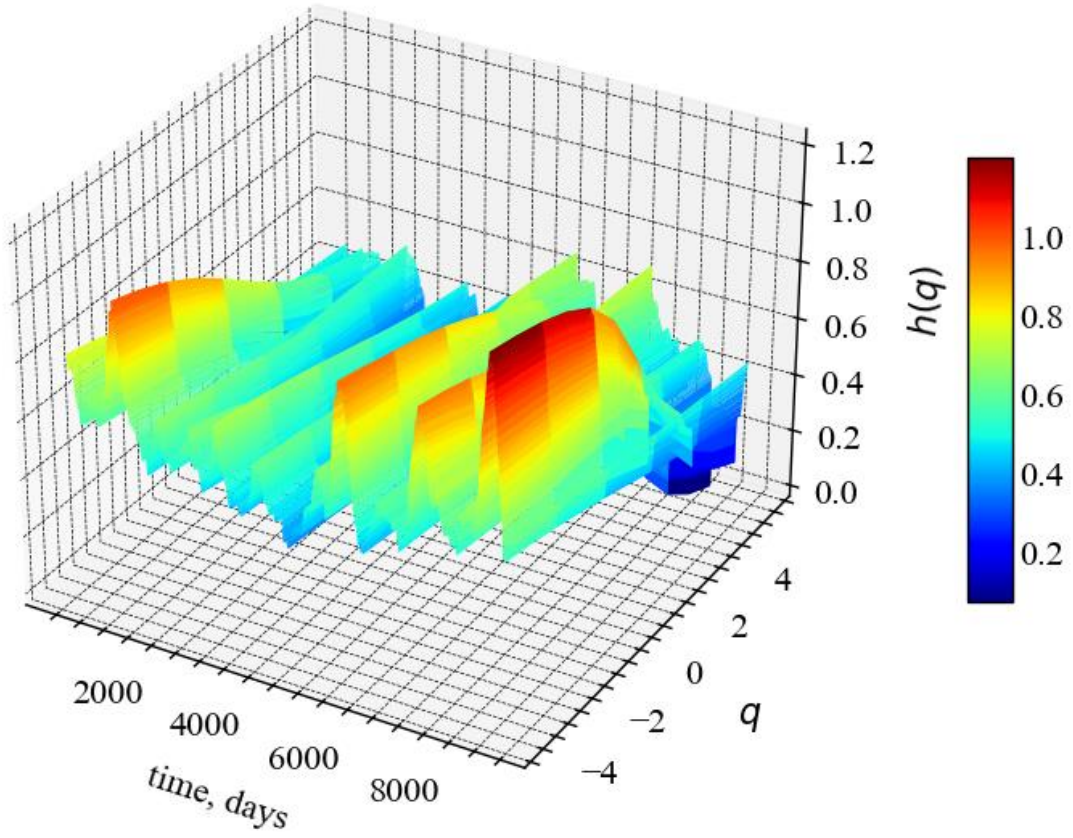


Рис. 7.39: Тривимірна діаграма динаміки узагальненого показника Херста $h(q)$, що змінюється з плином часу

На представлених рисунках (Рис. 7.38 та Рис. 7.39) видно, що узагальнений показник Херста характеризується значним ростом саме в період криз. Особливо високим $h(q)$ предстає для $q < 0$, що говорить про значну персистентність малих флуктуацій у періоди турбулентності. Найвищим ступінь нелінійності в даному випадку представляють кризи 1992, 2008-2009, 2015-2016 та 2020-2021 років, що підтверджується й попередніми індикаторами.

7.2.14.2 Динаміка $\tau(q)$ з ходом часу в дво- та тривимірному просторах

```
X, Y = np.meshgrid(time_ser.index[window:length:tstep], nq)
Z = np.array(tau_q).T

plot_2d(X, Y, Z,
        subtitle_jpg='contour_tau(q)',
        subtitle_fig=fr"Теплова діаграма $\tau(q)$",
        ylabel=r"$q$",
        barlabel=r"$\tau(q)$",
        cmap='viridis',
        lims=nq)
```

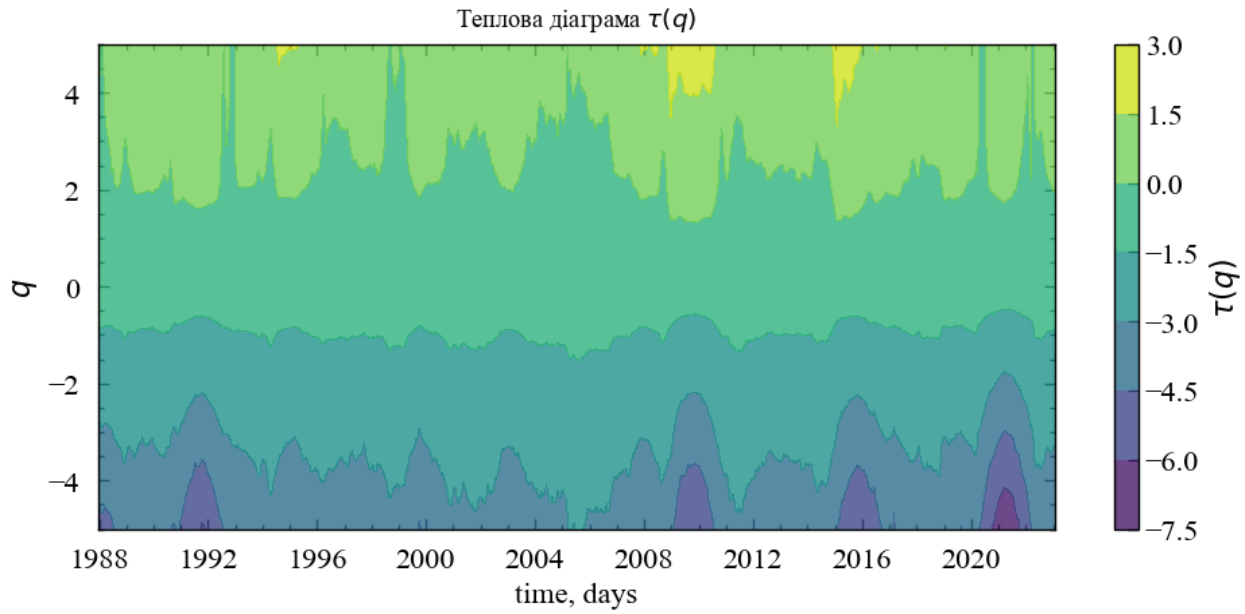


Рис. 7.40: Двовимірна контурна діаграма динаміки показника $\tau(q)$, що змінюється з плином часу

```
X, Y = np.meshgrid(np.arange(window, length, tstep), nq)
Z = np.array(tau_q).T

plot_3d(X, Y, Z,
        subtitle_jpg='3d_tau(q)',
        ylabel=r"$q$",
        zlabel=r"$\tau(q)$",
        cmap='viridis')
```

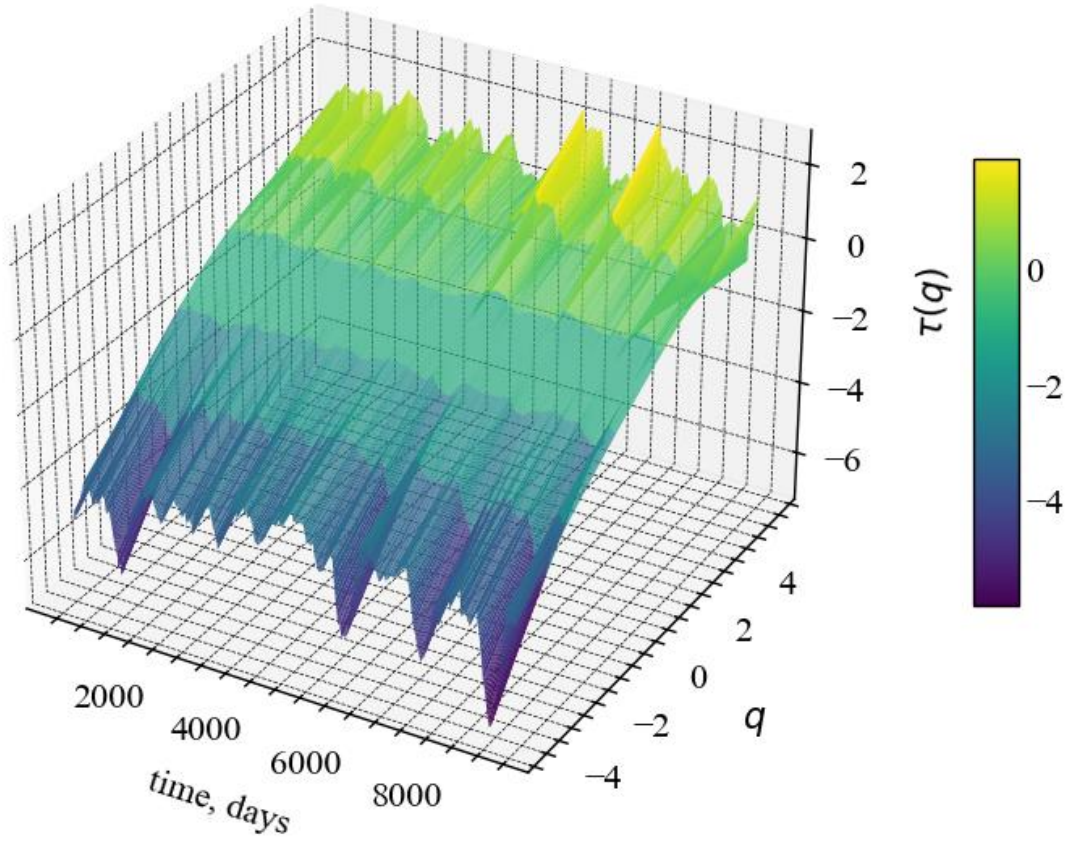


Рис. 7.41: Тривимірна діаграма динаміки показника $\tau(q)$, що змінюється з плином часу

Як видно з представлених рисунків (Рис. 7.40 та Рис. 7.41), $\tau(q)$ стає більш нелінійним для всіх значень q. На кінцях хвостів цього індикатора можна помітити значні впадини, що можуть слугувати індикаторами крахових подій, але в порівнянні з тим же показником Херста даний індикатор є менш виразним.

7.2.14.3 Динаміка $D(q)$ з ходом часу в дво- та тривимірному просторах

```
X, Y = np.meshgrid(time_ser.index[window:length:tstep], nq)
Z = np.array(D_q).T

plot_2d(X, Y, Z,
        subtitle_jpg='contour_D(q)',
        subtitle_fig=fr"Теплова діаграма $D(q)$",
        ylabel=r"$q$",
        barlabel=r"$D(q)$",
        cmap='magma',
        lims=nq)
```

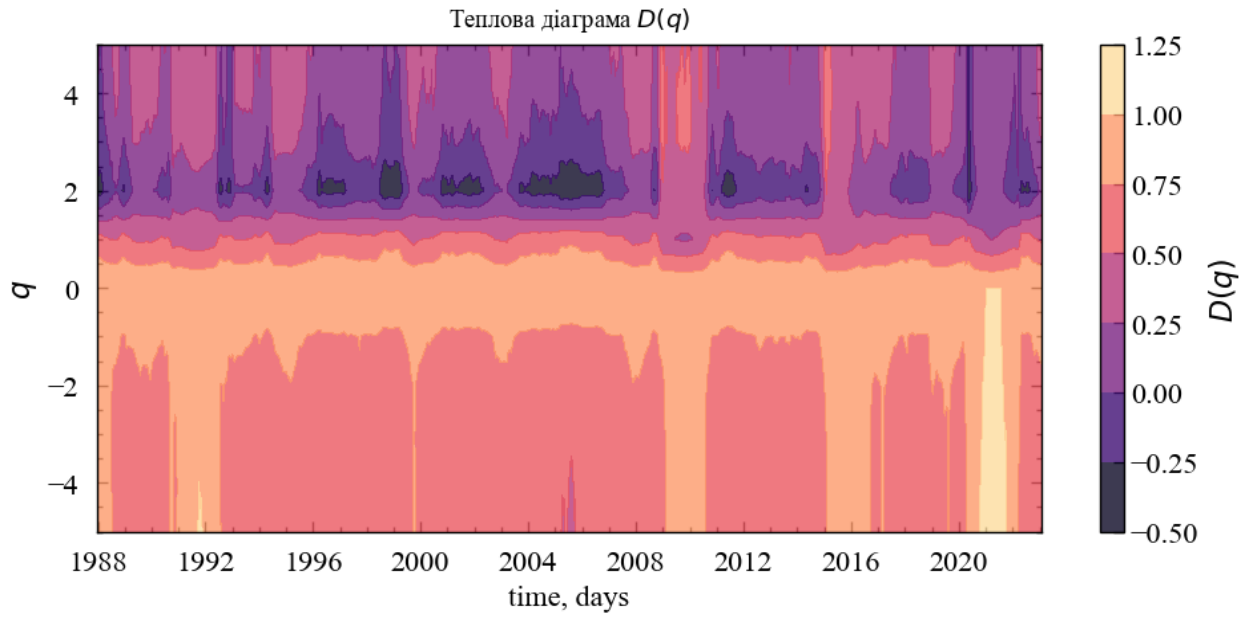


Рис. 7.42: Двовимірна контурна діаграма динаміки узагальненої фрактальної розмірності $D(q)$, що змінюється з плином часу

```
X, Y = np.meshgrid(np.arange(window, length, tstep), nq)
Z = np.array(D_q).T

plot_3d(X, Y, Z,
        subtitle_jpg='3d_D(q)',
        ylabel=r"$q$",
        zlabel=r"$D(q)$",
        cmap='magma')
```

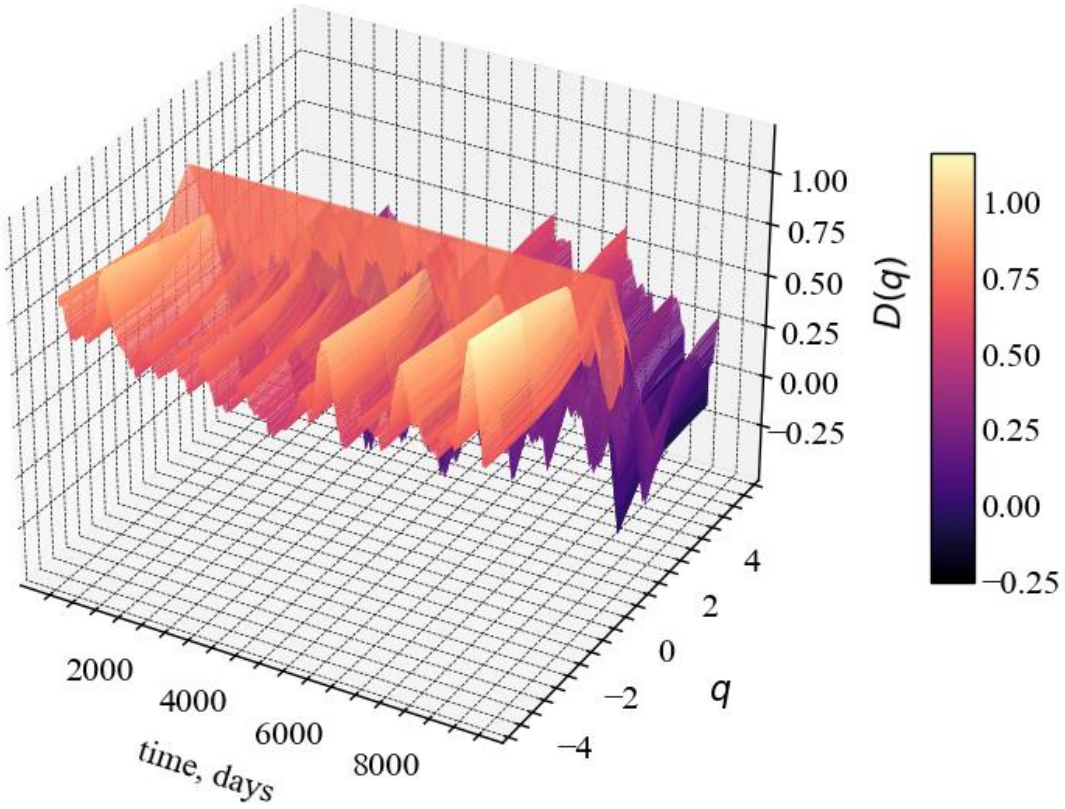


Рис. 7.43: Тривимірна діаграма динаміки узагальненої фрактальної розмірності $D(q)$, що змінюється з плином часу

Дво- та тривимірні представлення узагальненої фрактальної розмірності показують, що $D(q)$ зростає під час кризових подій. Узагальнена фрактальна розмірність також представляє найбільш індикативну динаміку для негативних значень q, хоча для позитивних q також спостерігаються незначні коливання.

7.2.14.4 Динаміка $C(q)$ в дво- та тривимірному просторах

```
X, Y = np.meshgrid(time_ser.index[window:length:tstep], nq)
Z = np.array(C_q).T

plot_2d(X, Y, Z,
        subtitle_jpg='contour_C(q)',
        subtitle_fig=fr"Теплова діаграма $C(q)$",
        ylabel=r"$q$",
        barlabel=r"$C(q)$",
        cmap='hot',
        lims=nq)
```

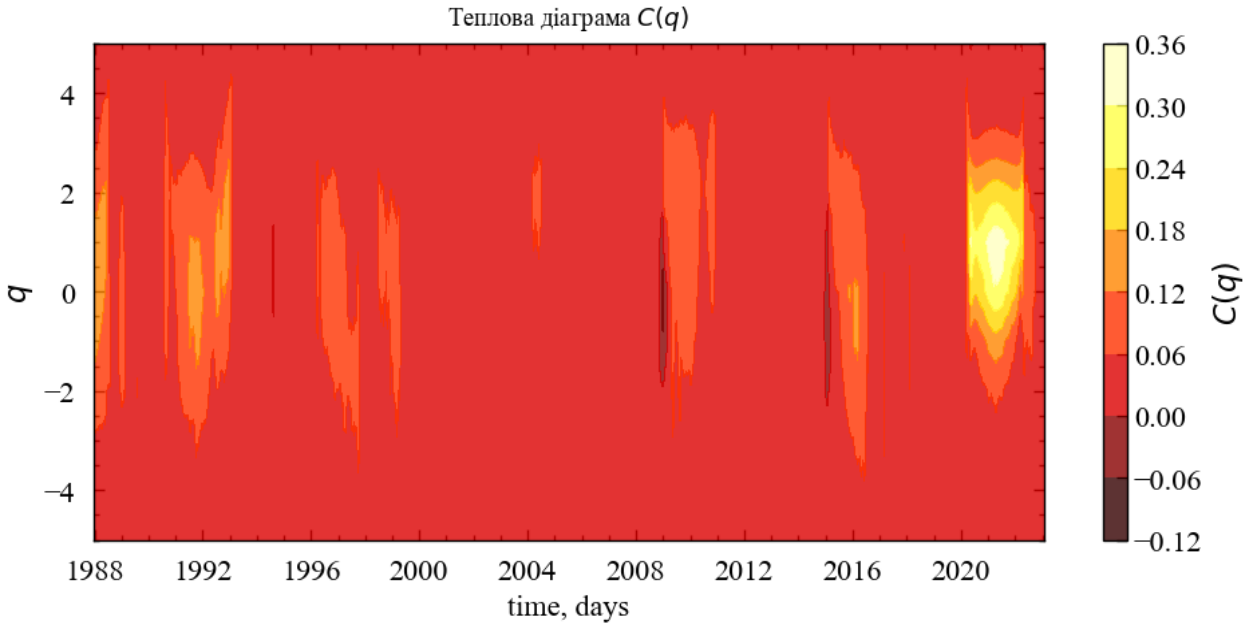


Рис. 7.44: Двовимірна контурна діаграма динаміки мультифрактальної теплоємності $C(q)$, що змінюється з плином часу

```
X, Y = np.meshgrid(np.arange(window, length, tstep), nq)
Z = np.array(C_q).T

plot_3d(X, Y, Z,
        subtitle_jpg='3d_C(q)',
        ylabel=r"$q$",
        zlabel=r"$C(q)$",
        cmap='hot')
```

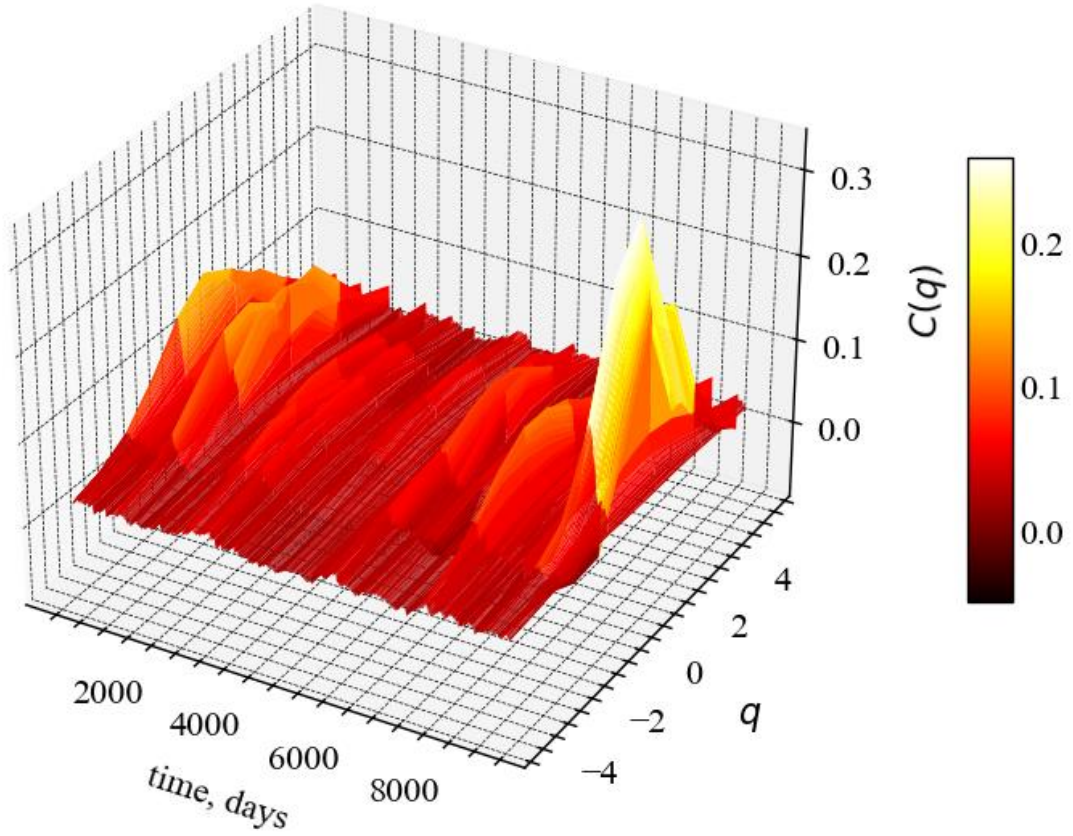


Рис. 7.45: Тривимірна контурна діаграма динаміки мультифрактальної теплоємності $C(q)$, що змінюється з плином часу

На даних рисунках (Рис. 7.44 та Рис. 7.45) під час кризових подій спостерігаються стрибки мультифрактальної теплоємності, що вказує аналогії фізичних фазових переходів та кризових подій. Можна бачити, що при різних ринкових режимах $C(q)$ може бути симетричною, демонструючи рівномірний вплив на динаміку ринку як висококонцентрованих елементів, так і низькоконцентрованих. Також $C(q)$ може зміщуватись як у ліву сторону, так і вправу, що говорить про мінливість ринку та впливовість різних початкових умов на його структуризацію.

7.2.14.5 Динаміка $f(\alpha)$ з ходом часу в дво- та тривимірному просторах

```
X = time_ser.index[window:length:tstep].values
X = np.expand_dims(X, axis=1)
X = np.repeat(a=X, repeats=nq.shape[0], axis=1)

Y = np.array(alpha)
Z = np.array(mfSpect)

plot_2d(X, Y, Z,
        subtitle_jpg='contour_f(alpha)',
```

```
        subtitle_fig=fr"Теплова діаграма $f(\alpha)$",
        ylabel=r"$\alpha$",
        barlabel=r"$f(\alpha)$",
        cmap='hsv',
        lims=alpha)
```

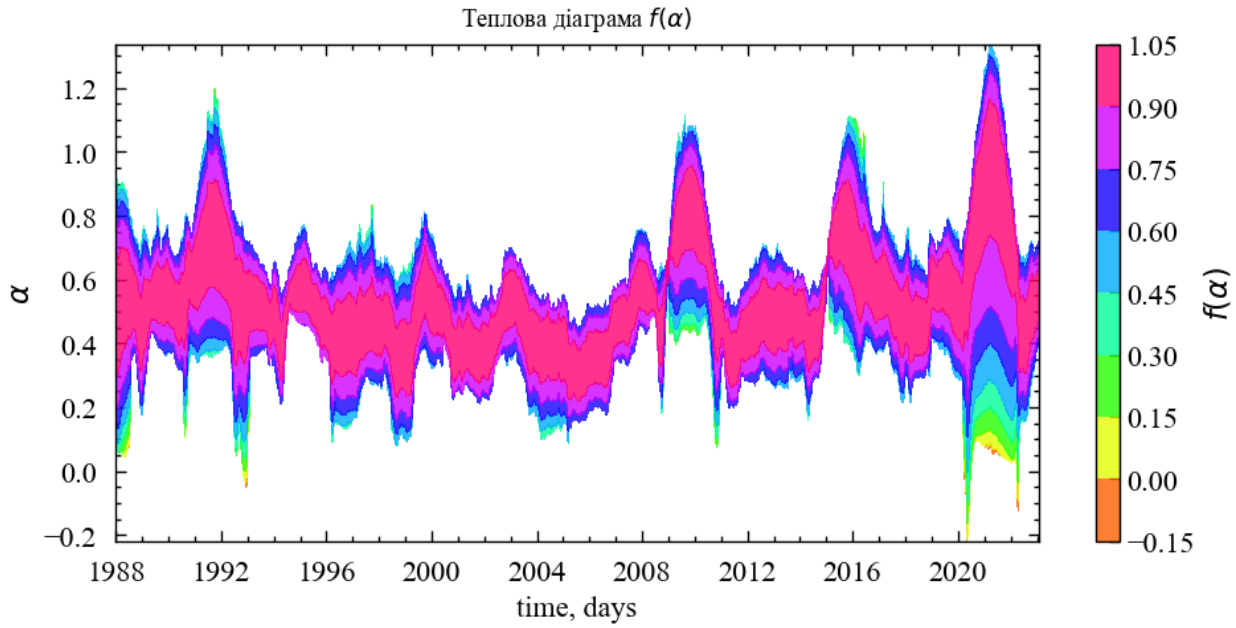


Рис. 7.46: Двовимірна контурна діаграма динаміки мультифрактального спектра $f(\alpha)$, що змінюється з плином часу

```
X = np.arange(window, length, tstep)
X = np.expand_dims(X, axis=1)
X = np.repeat(a=X, repeats=nq.shape[0], axis=1)

Y = np.array(alpha)
Z = np.array(mfSpect)

plot_3d(X, Y, Z,
        subtitle_jpg='3d_f(alpha)',
        ylabel=r"$\alpha$",
        zlabel=r"$f(\alpha)$",
        cmap='hsv')
```

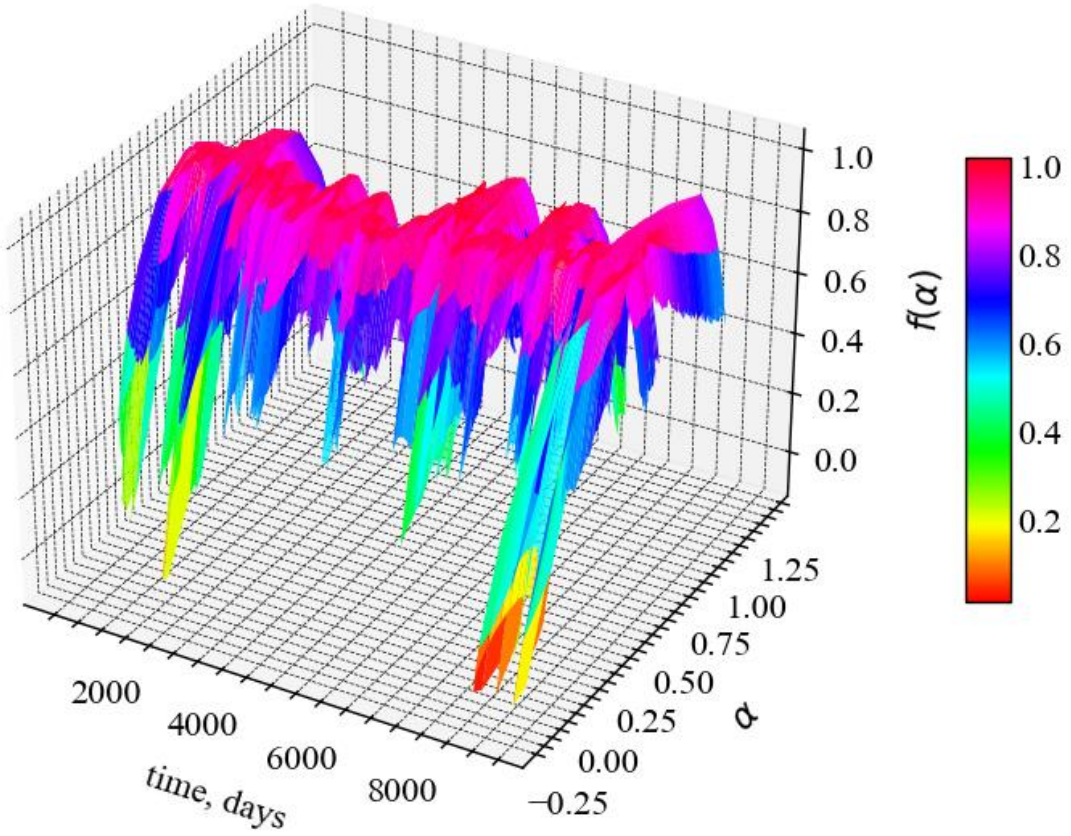


Рис. 7.47: Тривимірна діаграма динаміки мультифрактального спектра $f(\alpha)$, що змінюється з плином часу

Як видно з останніх рисунків (Рис. 7.46 та Рис. 7.47), ширина спектра мультифрактальності змінюється у формі з плином часу, і стає ширшою під час кризових подій, що підтверджувалось таким індикатором як, наприклад, $\Delta\alpha$. Видно, що у передкризові періоди зростає лівостороння асиметрія, що характеризує флуктуації значної амплітуди коливань. Самі кризи представляють зміщення $f(\alpha)$ у праву сторону, що вказує на домінацію флуктуацій з малою амплітудою. У будь-якому разі, зростання ширини спектра є індикатором зростання ступеня самоорганізованості елементів, що залучені до досліджуваної системи. Тобто, як $f(\alpha)$, так і попередні індикатори можна рекомендувати в якості індикаторів або індикаторів-передвісників кризових подій. У подальших лабораторних роботах цікаво буде розглянути різновиди MF-DFA, що, наприклад, враховують мультифрактальні крос-кореляції [126].

7.3 Висновок

У даній лабораторній роботі було представлено спектр мультифрактальних показників у якості індикаторів (індикаторів-передвісників) крахових подій.

Було показано, що відповідні показники поводять себе характеристичним чином (зростають чи спадають) у кризові та передкризові періоди на ринку нафти. По аналогії з лаб. 6 можна бачити, що ринок нафти характризується варіативністю ступеня мультифрактальності, що говорить про зміну кореляцій як малих флуктуацій, так і великих на різних просторово-часових шкалах. Подальші дослідження можуть бути спрямовані на вивчення можливості визначення порогових значень ступеня мультифрактальності, які можна було б використовувати для визначення ступеня розвинутості фінансових ринків. Деякі ринки, що розвиваються, можуть бути більш розвиненими, ніж інші, оскільки вони мають динамічні та зростаючі, а тому їх спектр мультифрактальності може бути найширшим. Одже, у цьому випадку можна визначати різні стадій розвитку ринку і моделювати змінни в динаміці складних систем як функцію ступеня мультифрактальності.

7.4 Завдання для виконання

1. Вибрати із запропонованої бази даних варіант завдання
2. Провести повний аналіз ряду за допомогою методу MF-DFA
3. Дати інтерпретацію отриманим результатам

7.5 Контрольні запитання

1. В чому переваги мультифрактальних характеристик у порівнянні з монофрактальними?
2. На що вказує мультифрактальність часового ряду?
3. Яким чином поводять себе різні характеристики, якщо ряд містить кризу?

8. Лабораторна робота № 8

Тема. Дослідження процесів самоорганізації в складних системах із використання теорії випадкових матриць

Мета. Навчитись використовувати методи теорії випадкових матриць для дослідження колективних процесів у складних системах

8.1 Теоретичні відомості

Вивчення статистичних властивостей матриць з незалежними випадковими елементами — випадкових матриць — має багату історію, що починається з ядерної фізики [127–133], де проблема з'явилася більше 50 років тому при дослідженні енергетичних рівнів складних ядер. **Теорія випадкової матриці** (ТВМ) була розвинена в цьому контексті Вігнером (Wigner), Дайсоном (Dyson), Метою (Mehta) та іншими для пояснення статистики рівнів енергії складних квантових систем. Дослідники постулювали, що функція Гамільтона, яка описує важкі ядра, може бути задана матрицею H з незалежними випадковими елементами H_{ij}, отриманими з розподілу імовірності. Відштовхуючись від цього припущення було зроблено низку вражаючих передбачень, які було підтверджено експериментально. Для складних квантових систем результати на основі ТВМ представляють середнє за всіма можливими взаємодіями. Відхилення від універсальних передбачень ТВМ відображують системну специфіку, невипадкові властивості системи, забезпечуючи ключові підходи до розуміння базової взаємодії системи. Недавні дослідження, що використовували методи аналізу ТВМ до аналізу властивостей матриці взаємних кореляцій C для економічних систем, показують, що близько 98% власних значень матриці C співпадають зі значеннями, отримуваними з використанням ТВМ, таким чином пропонуючи задовільний рівень хаотичності у вимірюваних крос-кореляціях. Також було знайдено, що існують відхилення від передбачень за допомогою ТВМ у близько 2% найбільших власних значень. Ці результати викликають наступні питання:

1. Яка можлива інтерпретація для відхилень від ТВМ?
2. Що можна сказати про структуру C з цих результатів?
3. Яке практичне значення отриманих результатів?

Шляхом комп'ютерного моделювання виявлено, що найбільше власне значення матриці C представляє вплив ринку в цілому. Аналіз змісту власних

значень, що відхиляються від ТВМ, показує існування взаємних кореляцій між акціями того ж самого типу діяльності, найбільш капіталізованими акціями, і акціями фірм, що мають бізнес у певному географічному секторі (локалізовані територіально). Обчислюючи скалярний добуток власних векторів від одного періоду часу до наступного, можна побачити, що власні вектори, що відхиляються, мають різні ступені стабільності в часі, визначеному кількісно величиною скалярного добутку. Найбільші два-три власних вектори стійкі протягом тривалих періодів часу, у той час як для іншої частини власних векторів, що відхиляються, стабільність у часі зменшується як тільки відповідні власні значення наближаються до верхньої межі ТВМ.

8.1.1 Знаходження коефіцієнтів матриці крос-кореляцій

Визначення кореляцій між різними акціями — тема, цікава тема не лише з точки зору розуміння економіки як складної динамічної системи, але також і прагматично, зокрема, з точки зору розміщення активів і оцінки портфельного ризику. Ми будемо аналізувати взаємні кореляції між акціями, застосовуючи поняття і методи теорії випадкових матриць, що використовуються в контексті складних квантових систем, де точний характер взаємодій між підодиницями невідомий.

Для визначення кількісно кореляцій спочатку обчислюється зміна цін (прибутковості) акції $i = 1, \dots, N$ за час Δt,

$$G_i(t) = \ln S_i\,(t + \Delta t) - \ln S_i(t), \tag{8.1}$$

де $S_i(t)$ позначає ціну акції i. Оскільки різні ціни мають різні рівні змінюванності (стандартні відхилення), визначатимемо стандартизовану прибутковість

$$g_i(t) \equiv [G_i(t) - \langle G_i \rangle]/\sigma_i, \tag{8.2}$$

де $\sigma_i \equiv \sqrt{\langle G_i^2 \rangle - \langle G_i \rangle^2}$ — стандартне відхилення G_i, а $\langle \dots \rangle$ позначає середнє значення за досліджуваний період часу. Тоді знаходження матриці кореляцій C зводиться до обчислення формули:

$$C_{ij} \equiv \langle g_i(t) g_j(t) \rangle. \tag{8.3}$$

За побудовою, елементи C_{ij} обмежені областю $-1 \leq C_{ij} \leq 1$, де $C_{ij} = 1$ відповідає повним кореляціям, $C_{ij} = -1$ — повним антикореляціям, і $C_{ij} = 0$ свідчить про некорельованність пар акцій.

Труднощі в аналізі важливості та значення коефіцієнтів крос-кореляції C_{ij} виникають внаслідок кількох причин, що полягають в наступному:

- ринкові умови з часом змінюються і взаємна кореляція, що існує між будь-якою парою акцій, може бути не постійною (нестаціонарною);
- скінчена довжина досліджуваного ряду, доступного для оцінювання взаємних кореляцій, додає так званий "шум вимірювання" — чим коротший досліджуваний ряд — тим менш точними будуть отримувані значення.

Яким же чином можна виділяти з C_{ij} ті акції, що залишилися корельованими на розглядуваному періоді часу? Щоб відповісти на це питання, перевіримо статистику C у порівнянні із так званою "нульовою гіпотезою" випадкової кореляційної матриці — матриці кореляцій, побудованої із взаємно некорельованих часових рядів. Якщо властивості C відповідають властивостям для випадкової матриці кореляцій, тоді можна говорити про те, що значення емпірично вимірюваних властивостей C випадкові. Навпаки, відхилення властивостей C від таких же властивостей для випадкової кореляційної матриці передає інформацію про "справжні" кореляції. Отже, нашою метою є порівняння властивостей C з такими ж властивостями випадкової матриці кореляцій і розділ властивостей C на дві групи: (a) частина C, що відповідає властивостям випадкової кореляційної матриці ("шум") і (b) частина C, що відхиляється ("інформація").

8.1.2 Розподіл власних значень

Для отримання інформації про взаємні кореляції C необхідно порівняти властивості C з такими ж властивостями випадкової матриці крос-кореляцій [134]. У матричній нотації така матриця може бути виражена як

$$C = \frac{1}{L} G G^T, \tag{8.4}$$

де G — матриця розміру $N \times L$ з елементами $g_{im} = g_i(m\Delta t), i = 1, \ldots, N; m = 0, \ldots, L-1$ і G^T позначає транспонування G. Розглянемо випадкову кореляційну матрицю

$$R = \frac{1}{L} AA^T, \tag{8.5}$$

де A — матриця розміру $N \times L$, що містить N часових рядів із L випадковими елементів a_{im} з нульовим середнім і одиничним відхиленням, що означають взаємну некорельованість. За побудовою R належить до типу матриць, які часто називають матрицями Вішарта у багатовимірній статистиці [135].

Статистичні властивості випадкових матриць R відомі [136,137]. Зокрема, у наближенні $N \to \infty$, $L \to \infty$, такому, що $Q \equiv L/N (> 1)$ фіксоване, показано аналітично [137], що функція розподілу щільності імовірності $P_{rm}(\lambda)$ власних значень λ випадкової матриці кореляції R визначається як

$$P_{rm}(\lambda) = \frac{Q}{2\pi} \frac{\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}}{\lambda}, \tag{8.6}$$

із λ в межах границь $\lambda_- \leq \lambda_i \leq \lambda_+$, де λ_- і λ_+ — найменше та найбільше власні значення R, які можна визначити аналітично як

$$\lambda_\pm = 1 + 1/Q \pm 2\sqrt{1/Q}. \tag{8.7}$$

Вираз (8.6) є точним для випадку розподілених за Гаусом матричних елементів a_{im}.

Порівняємо розподіл власних значень $P(\lambda)$ для C з $P_{rm}(\lambda)$. Для цього обчислимо власні значення λ_i матриці C, причому λ_i впорядкуємо за зростанням ($\lambda_{i+1} > \lambda_i$). При дослідженнях зверніть увагу на присутність чіткої “великої частини” власних значень, що спадають у межах границь $[\lambda_-, \lambda_+]$ для $P_{rm}(\lambda)$. Також зверніть увагу на відхилення для деяких найбільших і найменших власних значень отриманих за допомогою ТВМ.

Оскільки (8.6) є таким, що строго відповідає лише для $L \to \infty$ і $N \to \infty$, необхідно перевірити також відхилення від ідеального випадку, оскільки робота проводиться завжди зі **скінченими** рядами. При дослідженнях виявляється, що для кількох найбільших (найменших) власних значень ефект впливу скінчених величин L і N відсутній.

8.1.3 Розподіл власних векторів

Відхилення $P(\lambda)$ від передбачення ТВМ $P_{rm}(\lambda)$ свідчить про те, що ці відхилення також повинні відображатися в статистиці відповідних компонент

власного вектора [134]. Відповідно, у даній лабораторній ми будемо аналізувати розподіл компоненти власного вектора. Розподіл компонент $\{u_l^k; l = 1, \ldots, N\}$ власного вектора u^k випадкової кореляційної матриці R має відповідати розподілу Гауса з нульовим середнім та одиничною дисперсією:

$$\rho_{rm}(u) = \frac{1}{\sqrt{2\pi}} \exp(-u^2/2).$$

8.1.4 Обернене відношення участі

Вивчивши інтерпретацію найбільшого власного значення, що значно відхиляється від результатів ТВМ, зосередимось на власних значеннях, що залишаються. Відхилення розподілу компонентів власного вектора u_k від ТВМ Гаусового передбачення більш явне, коли відстань від верхньої границі ТВМ $\lambda_k - \lambda_+$ збільшується. Оскільки близькість до λ_+ збільшує ефекти хаотичності, визначаємо кількість компонентів, що беруть значну участь у кожному власному векторі, що, у свою чергу, відображає ступінь відхилення від ТВМ для розподілу компонентів власного вектора. Для цього використовується поняття **оберненого відношення участі** (ОВУ) [138–140], що часто застосовується в теорії локалізації. ОВУ власного вектора u_k визначається як

$$I^k \equiv \sum_{l=1}^{N} \left[u_l^k\right]^4, \tag{8.8}$$

де u_l^k, $l = 1, \ldots, N$ — компоненти власного вектора u^k. Значення I^k може бути проілюстровано двома граничними випадками:

- вектор з ідентичними компонентами $u_l^k \equiv 1/\sqrt{N}$ має $I^k = 1/N$;
- вектор з одним компонентом $u_1^k = 1$ і нульовими іншими має $I^k = 1$.

Таким чином, ОВУ визначає кількість даних з числа компонентів власного вектора, що значно впливають на ринок, заданий системою часових рядів. Наявність векторів з великими значеннями I^k також виникає в теорії локалізації Андерсона. У контексті теорії локалізації часто знаходять “випадкову смугу матриць”, що містять узагальнені стани з маленьким I^k в більшій частині спектра власних значень, тоді як основні стани локалізовані і мають великі I^k. Виявлення локалізованих станів для маленьких і великих власних значень матриці крос-кореляцій C нагадує про локалізацію Андерсона і припускає, що C може мати випадкову зону матричної структури.

8.2 Хід роботи

Імпортуємо необхідні бібліотеки:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib.ticker as mticker
import pandas as pd
import yfinance as yf
import scienceplots
import requests
from tqdm import tqdm

np.seterr(divide='ignore', invalid='ignore')

%matplotlib inline
```

Визначаємо стиль рисунків:

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
'figure.figsize': (8, 6),           # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                  # розмір фонтів рисунку
'lines.linewidth': 2,               # товщина ліній
'axes.titlesize': 'small',          # розмір титулки над рисунком
'axes.labelsize': size,             # розмір підписів по осям
'legend.fontsize': size,            # розмір легенди
'xtick.labelsize': size,            # розмір розмітки по осі Ox
'ytick.labelsize': size,            # розмір розмітки по осі Ox
"font.family": "Serif",             # сімейство стилів підписів
"font.serif": ["Times New Roman"],  # стиль підпису
'savefig.dpi': 300,                 # якість збережених зображень
'axes.grid': False                  # побудова сітки на самому рисунку
}

plt.rcParams.update(params)         # оновлення стилю згідно налаштувань
```

Виконуємо парсинг та фільтрацію заголовків акцій компаній:

```
headers = {
'authority': 'api.nasdaq.com',
'accept': 'application/json, text/plain, */*',
'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/87.0.4280.141 Safari/537.36',
'origin': 'https://www.nasdaq.com',
'sec-fetch-site': 'same-site',
'sec-fetch-mode': 'cors',
'sec-fetch-dest': 'empty',
'referer': 'https://www.nasdaq.com/',
'accept-language': 'en-US,en;q=0.9',
}
```

```
params = (
    ('tableonly', 'true'),
    ('limit', '25'),
    ('offset', '0'),
    ('download', 'true'),
)

r = requests.get('https://api.nasdaq.com/api/screener/stocks', headers=headers,
params=params)
data = r.json()['data']
df = pd.DataFrame(data['rows'], columns=data['headers'])
df = df.dropna(subset={'marketCap'})
df = df[~df['symbol'].str.contains("\/|\.|\^")]
```

Фільтруємо та сортуємо заголовків акцій за їх капіталізацією:

```
def cust_filter(mkt_cap):
    if 'M' in mkt_cap:
       return float(mkt_cap[1:-1])
    elif 'B' in mkt_cap:
       return float(mkt_cap[1:-1]) * 1000
    elif mkt_cap == '':
       return 0.0
    else:
       return float(mkt_cap[1:]) / 1e6

df['marketCap'] = df['marketCap'].apply(cust_filter)
df = df.sort_values('marketCap', ascending=False)
```

Визначаємо найпередовіші акцій за їх капіталізацією:

```
top = 150
tickers_list = df.iloc[:top]['symbol'].tolist()
tickers_list[:10]

['GOOG', 'AMZN', 'GOOGL', 'AAPL', 'NVDA', 'META', 'MSFT', 'CRM', 'NVS', 'TMUS']
```

Зчитуємо дані з Yahoo Finance згідно створенного списку акцій. Вилучимо ціни закриття всіх акцій за період з 31 грудня 2001 року по 1 жовтня 2023 року:

```
start = "2001-12-31"
end = "2023-12-01"
data_init = yf.download(tickers_list, start, end)["Adj Close"]

xlabel = 'time, days'# підпис по вісі Ox
```

Далі нам потребується обрати фінансовий індекс для порівняння з розрахованими індикаторами. Далі надається список усіх доступних для нас акцій:

```
ylabel = 'AAPL' # підпис по вісі Oy

data_init.columns.values

array(['AAPL', 'ABBV', 'ABT', … ,'WMT', 'WSM', 'XOM'], dtype=object)
```

```
perc = 5.0
min_count = int(((100-perc)/100)*data_init.shape[0] +1)
data_init = data_init.dropna(axis=1, thresh=min_count)
data_init

data_init = data_init.dropna(axis=1)

data = data_init.T
```

8.2.1 Знаходження коефіцієнтів матриці крос-кореляцій

Перш за все нам потребується процедура для перетворення ряду до потрібного нам виду. Для цього визначимо функцію `transformation()`. З її допомогою для всього ряду та у віконній процедурі будемо рахувати прибутковості та всі необхідні індикатори.

```
def transformation(signal, ret_type):

    for_rmt = signal.copy()
    # Зважаючи на вид ряду, виконуємо
    # необхідні перетворення
    if ret_type == 1:
        for_rmt.values
    elif ret_type == 2:
        for_rmt = for_rmt.diff(axis=1).values
    elif ret_type == 3:
        for_rmt = for_rmt.pct_change(axis=1).values
    elif ret_type == 4:
        for_rmt = for_rmt.pct_change(axis=1).dropna(axis=1).values
        for_rmt -= np.nanmean(for_rmt, axis=1, keepdims=True)
        for_rmt /= np.nanstd(for_rmt, axis=1, keepdims=True)
    elif ret_type == 5:
        for_rmt = for_rmt.pct_change(axis=1).values
        for_rmt = for_rmt[~np.isnan(for_rmt).any(axis=1)]
        for_rmt -= np.mean(for_rmt, axis=1, keepdims=True)
        for_rmt /= np.std(for_rmt, axis=1, keepdims=True)
        for_rmt = np.abs(for_rmt)
    elif ret_type == 6:
        for_rmt = for_rmt.values
        for_rmt -= np.mean(for_rmt, axis=1, keepdims=True)
        for_rmt /= np.std(for_rmt, axis=1, keepdims=True)

    return for_rmt

type = 4

log_ret = transformation(data, type)

N, T = log_ret.shape
```

Будуємо матрицю кореляцій для матриці прибутковостей наших активів:

```
def calc_cross_corr(data):
    C = (1/T)*np.dot(data, data.T)
    di = np.diag_indices(data.shape[0])
```

```
    ccoef = np.ma.asarray(C)
    ccoef[di] = np.ma.masked
    ccoef_flat = ccoef.compressed()

    return C, ccoef_flat

C, ccoef_flat = calc_cross_corr(log_ret)
```

І матрицю кореляцій для перемішаної матриці прибутковостей:

```
np.random.seed(1234)
random_stocks = np.random.normal(size=(N,T))

R, ccoef_flat_rand = calc_cross_corr(random_stocks)
```

Виводимо результат:

```
fig = plt.figure(figsize=(8, 6))
gs = gridspec.GridSpec(2, 2)

ax1 = fig.add_subplot(gs[0, 0])
im1 = ax1.imshow(C, cmap='hot', interpolation='nearest')
ax1.invert_yaxis()
fig.colorbar(im1, ax=ax1)

ax2 = fig.add_subplot(gs[0, 1])
im2 = ax2.imshow(R, cmap='hot', interpolation='nearest')
ax2.invert_yaxis()
fig.colorbar(im2, ax=ax2)

ax3 = fig.add_subplot(gs[1, :])
ax3.hist(ccoef_flat, bins='auto', density=True, label=r'$P^{init}(C_{ij})$')
ax3.hist(ccoef_flat_rand, bins='auto', density=True,
label=r'$P^{rand}(C_{ij})$')
ax3.set_xlabel(r'$C_{ij}$')
ax3.set_ylabel(r'$P(C_{ij})$')
ax3.legend()

fig.align_labels()
plt.show();
```

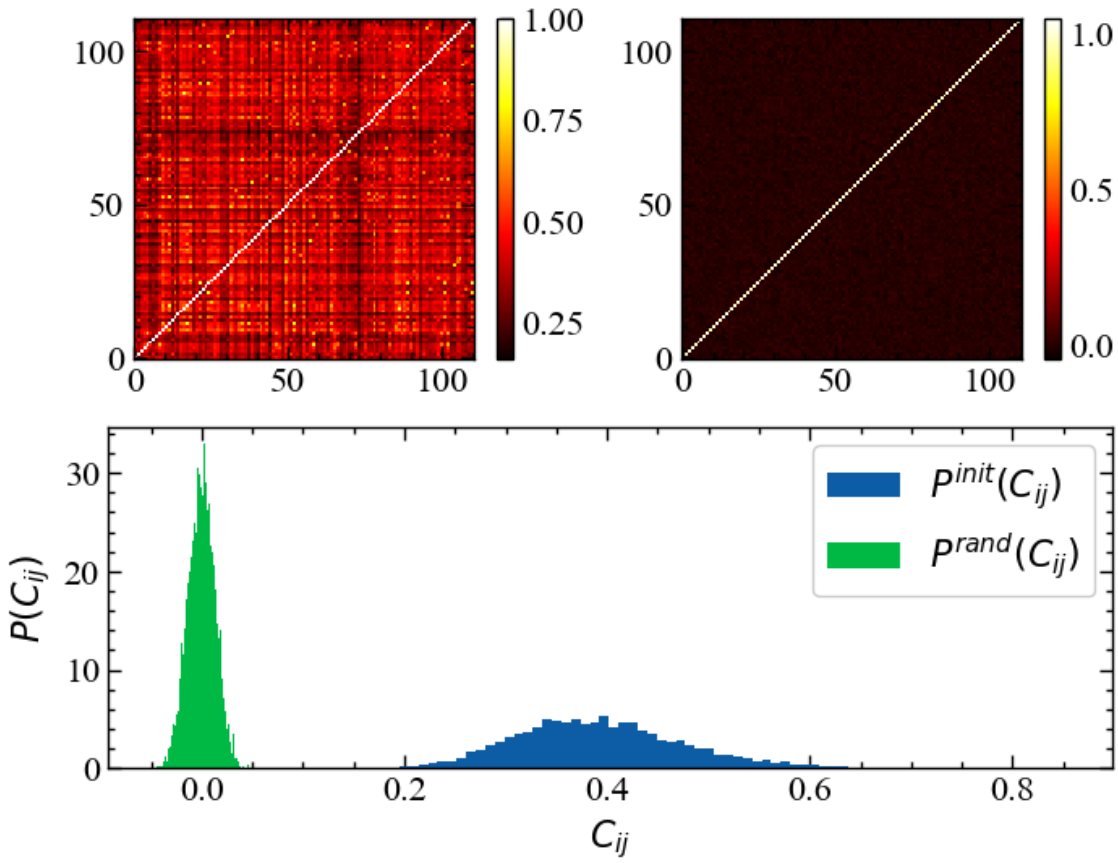


Рис. 8.1: Карти полів кореляцій вихідної і перемішаної матриць. У нижній частині рисунку представлено порівняння розподілів значень коефіцієнтів кореляції вихідної та перемішаної матриць

На Рис. 8.1 видно, що розподіл коефіцієнтів кореляції для вихідної матриці значно відхиляється розподілу для випадкової матриці. Можна помітити, що $P^{init}(C_{ij})$ представляється доволі розтягненим в діапазоні $C_{ij} \in [0.1,0.8]$, що вказує на те, що достатньо багато досліджуваних активів досить сильно відрізняють один від одного. Можна виокремити активи, які характеризуються досить значним ступенем лінійної залежності по відношенню до більності активів на протязі значного часу існування, а є такі активи, які проявляють лінійну залежність лише в конкретних ринкових умовах. Тим не менш, увесь ринок характеризується досить значним ступенем детермінованості.

8.2.2 Розподіл власних значень та векторів

```
def calc_lambd_eig(C):
return np.linalg.eig(C)

lambdas, u = calc_lambd_eig(C)
lambdas_rand, u_rand = calc_lambd_eig(R)

Q = T/N
```

```
lambda_plus = 1+1/Q+2*np.sqrt(1/Q)
lambda_minus = 1-1/Q-2*np.sqrt(1/Q)

print("Верхня границя розподілу власних значень, що прогнозується ТВМ: ",
lambda_plus)
print("Нижня границя розподілу власних значень, що прогнозується ТВМ: ",
lambda_minus)
```

```
Верхня границя розподілу власних значень, що прогнозується ТВМ:
1.3038069950128888
Нижня границя розподілу власних значень, що прогнозується ТВМ:
0.6961930049871112
```

Генеруємо 1000 випадкових значень в діапазоні від λ_- до λ_+, і генеруємо $P_{rm}(\lambda)$.

```
lambda_random = np.linspace(lambda_minus, lambda_plus, 1000)
P_rm = (Q/(2*np.pi))*np.sqrt((lambda_plus-lambda_random)*(lambda_random-
lambda_minus))/lambda_random
```

Виводимо результат:

```
fig, ax = plt.subplots(2, 1, figsize=(8, 6))
ax[0].hist(lambdas, bins=50, density=True, color="skyblue",
label='$P^{init}_{\it{empiric}}$')
ax[0].hist(lambdas_rand, bins='auto', density=True, color="red",
label='$P^{rand}_{\it{empiric}}$')
ax[0].set_xlabel('$\lambda_{i}$')
ax[0].set_ylabel('$P(\lambda_{i})$')
ax[0].text(20, 0.5, '$\lambda_{max}=$'+f'{lambdas.max():.2f}', ha='center',
va='center')
ax[0].set_yscale('log')
ax[0].legend()

ax[1].hist(lambdas, bins='auto', density=True, color="skyblue",
label='$P^{init}_{\it{empiric}}$')
ax[1].hist(lambdas_rand, bins='auto', density=True, color="red",
label='$P^{rand}_{\it{empiric}}$')
ax[1].set_xlabel('$\lambda_{i}$')
ax[1].set_ylabel('$P(\lambda_{i})$')
ax[1].set_xlim(lambda_minus-3*np.std(lambdas_rand),
lambda_plus+3*np.std(lambdas_rand))
ax[1].scatter(lambda_random, P_rm, label='$P_{\it{RMT}}$', color='green')
ax[1].legend()

fig.tight_layout()
plt.show();
```

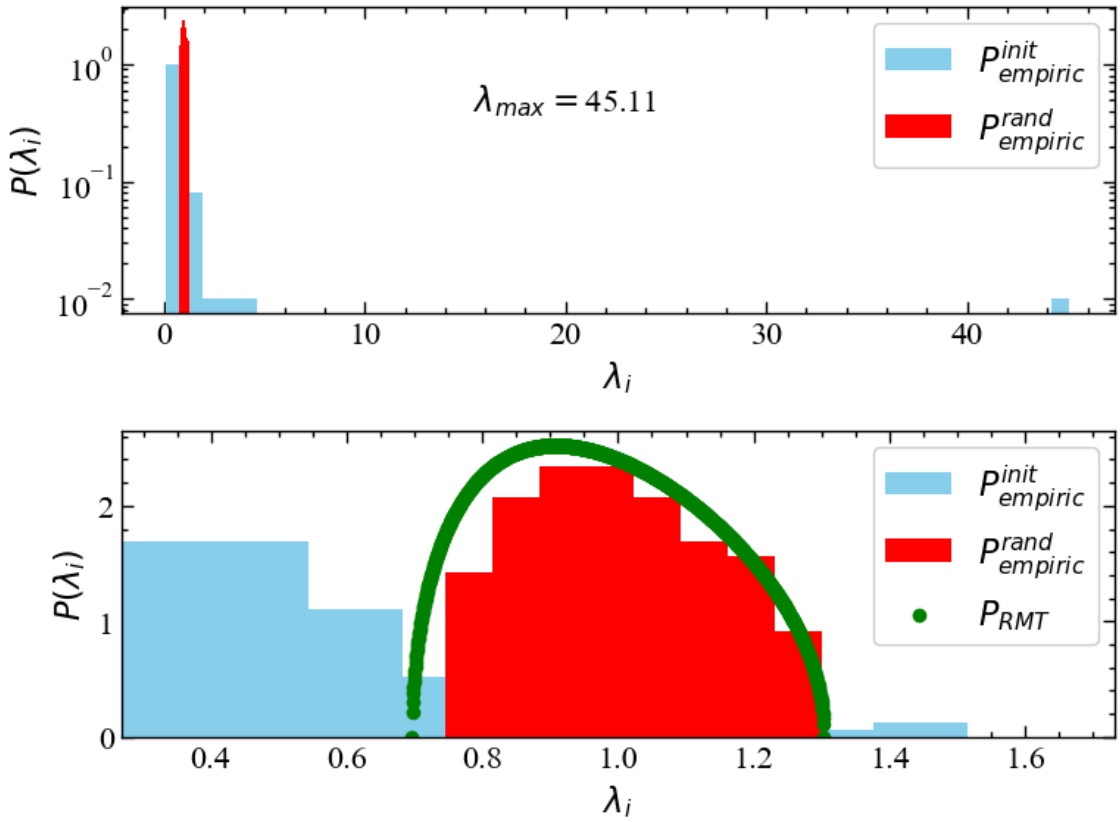


Рис. 8.2: Щільність розподілу ймовірностей для матриці прибутковостей фінансових активів $P^{init}_{empiric}$ та випадкової матриці $P^{rand}_{empiric}$. Другий рисунок представляє щільність розподілу, що прогнозується ТВМ (P_{RMT})

На Рис. 8.2 видно, що розподіл λ_i значно відхиляється від тієї частки, що відповідає випадковій матриці крос-кореляцій ($P^{rand}_{empiric}$). Для вихідного розподілу видно, що λ_{max} значно відхиляється від передбачення ТВМ. Для $P^{init}_{empiric}$ знаходяться і такі власні значення, що менші за ті, що знаходяться на нижній границі випадкової матриці. Загалом, така закономірність вказує на те, що на ринку є один або декілька індексів, що регулюють динаміку всього ринку.

На Рис. 8.3 представлено щільність розподілу ймовірностей компонент власних векторів вихідної матриці прибутковостей і випадкової матриці при λ_1, λ_{30} та λ_{70}.

```
fig, ax = plt.subplots(3, 1, figsize=(8, 6), sharex=True, sharey=True)
ax[0].hist(u[:, 0], bins=50, density=True, label=r'$\rho(u)_{empiric}^{init}$')
ax[0].hist(u_rand[:, 0], bins=50, density=True, alpha=0.7,
label=r'$\rho(u)_{empiric}^{rand}$')
ax[0].set_xlabel('$u_{1}$')
ax[0].set_ylabel('$P(u)$')
ax[0].set_title('Компоненти власного вектора при $ \lambda_{1} $')
ax[0].legend()
ax[0].set_yscale('log')

ax[1].hist(u[:, 30], bins=50, density=True, label=r'$\rho(u)_{empiric}^{init}$')
```

```
ax[1].hist(u_rand[:, 30], bins=50, density=True, alpha=0.7,
label=r'$\rho(u)_{empiric}^{rand}$')
ax[1].set_xlabel('$u_{30}$')
ax[1].set_ylabel('$P(u)$')
ax[1].set_title('Компоненти власного вектора при $ \lambda_{30} $')
ax[1].legend()
ax[1].set_yscale('log')

ax[2].hist(u[:, 70], bins=50, density=True, label=r'$\rho(u)_{empiric}^{init}$')
ax[2].hist(u_rand[:, 70], bins=50, density=True, alpha=0.7,
label=r'$\rho(u)_{empiric}^{rand}$')
ax[2].set_xlabel('$u_{70}$')
ax[2].set_ylabel('$P(u)$')
ax[2].set_title('Компоненти власного вектора при $ \lambda_{70} $')
ax[2].legend()
ax[2].set_yscale('log')

fig.tight_layout()
plt.show();
```

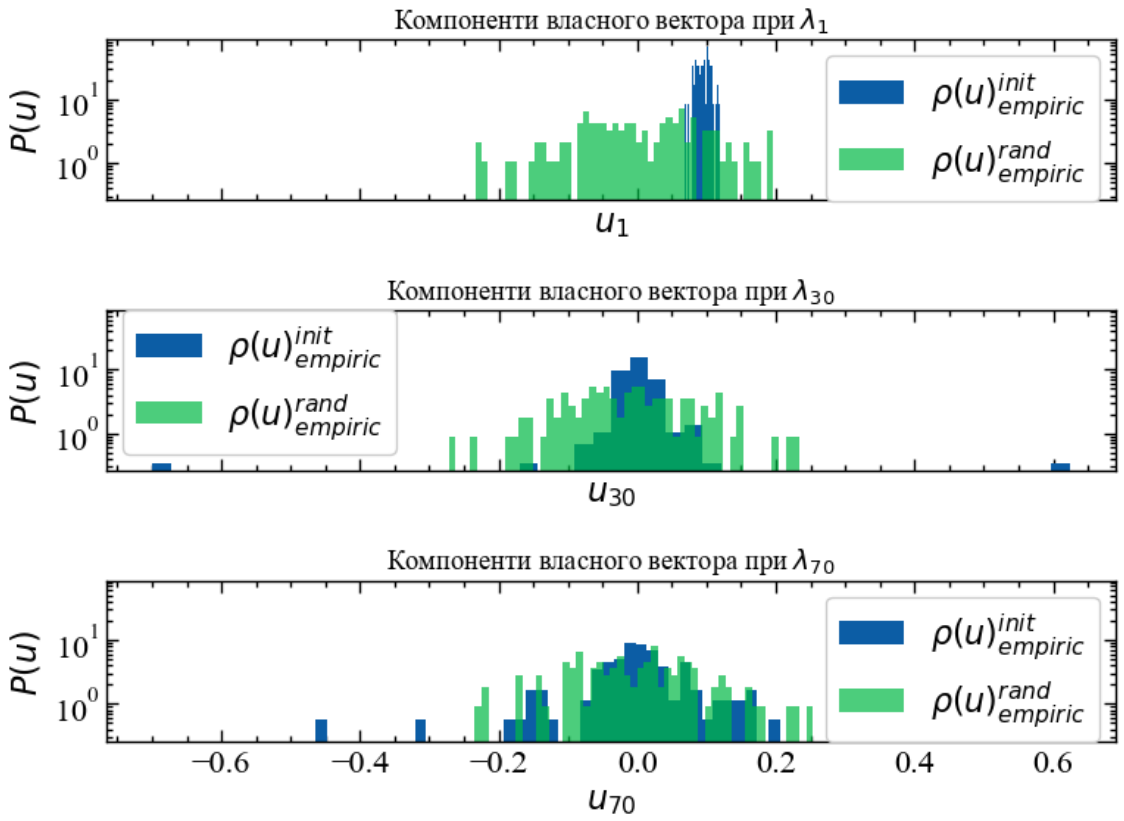


Рис. 8.3: Щільність розподілу ймовірностей компонент власних векторів вихідної матриці прибутковостей і випадкової матриці при λ_1, λ_{30} та λ_{70}

На Рис. 8.3 видно, що компоненти власного вектора при найбільшому власному значень значно відхиляються від передбачення ТВМ, що вказує на значний ступінь впливовості індексу, який відповідає λ_1. При λ_{30} та λ_{70} розподіл компонент власних векторів вихідних значень починає збігатися до розподілу

випадкових значень, що говорить про незначний вплив індексів, яким відповідають ці власні значення. Оскільки розподіл компонент їх векторів близький до ТВМ, можна сказати, що вони вносять найбільший шум у динаміку ринку.

На Рис. 8.4 представлено ОВУ для вихідної матриці крос-кореляцій та випадкової.

8.2.3 Обернене відношення участі

```
def calc_ipr(u):
    return np.sum(u**4, axis=0)

IPR = calc_ipr(u)
IPR_rand = calc_ipr(u_rand)

fig, ax = plt.subplots(1, 1)

ax.scatter(lambdas, IPR, color='green', label=r'$IPR_{init}$', s=10**2,
marker='x')
ax.scatter(lambdas_rand, IPR_rand, color='red', label=r'$IPR_{rand}$', s=10**2,
marker='x')
ax.set_xlabel(r'$\lambda_{i}$')
ax.set_ylabel(r'$IPR$')
ax.set_xscale('log')
ax.legend()

fig.tight_layout()
plt.show();
```

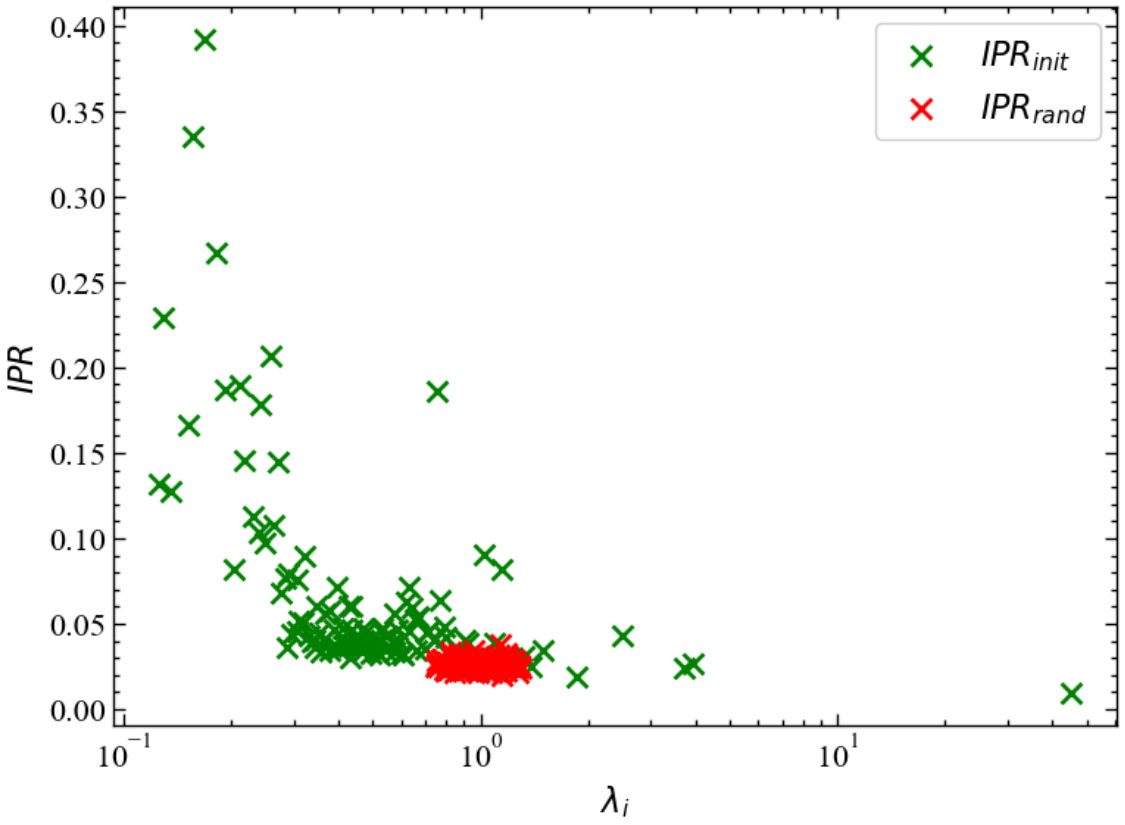


Рис. 8.4: ОВУ для вихідної та випадкової матриць крос-кореляцій

На Рис. 8.4 видно, що ОВУ для, наприклад, випадкової матриці концентрується в межах $\lambda_i \approx 10^0$. Для даних, що значно відхиляються від випадкових, ОВУ має довгі хвости, які виходять далеко за межі передбачень ТВМ. Для набільшого власного вектора ОВУ локалізується далеко в правому хвості розподілу. Також видно, що деякі власні значення мають $IRP \approx 0.25$. Це говорить про те, що компоненти деяких векторів розподілені асиметрично, що може вказувати на деяких ступінь впливу фондових індексів, яким властиві дані вектори.

8.2.4 Коефіцієнт поглинання

Цікаво буде буде порівняти максимальне власне значення λ_{max} з середнім значенням коефіцієнта кореляції і так званим **коефіцієнтом поглинання** (absorption ratio, AR), який є кумулятивною мірою ризику:

$$AR = \sum_{k=1}^{n} \lambda_k \Big/ \sum_{k=1}^{N} \lambda_k. \tag{8.9}$$

AR вказує, яку частину загальної варіації описують n із загальної кількості N власних значень.

Щоб вирішити, який власний вектор можна відкинути для розрахунків у чисельнику формули вище, не втрачаючи занадто багато інформації, нам потрібно перевірити відповідні власні значення: власні вектори з найменшими власними значеннями є найменш інформативними, тому їх і можна відсікти.

Загальний підхід полягає в упорядкуванні власних значень від найвищого до найнижчого.

Після сортування власних значень постає питання: “яку кількість найбільш інформативних власних компонент треба вибрати для розрахунку коефіцієнту поглинання?”. Корисною мірою є так звана “врахована (пояснена) дисперсія”, яка може бути обчислена за власними значеннями. Пояснена дисперсія говорить нам, яку частину інформації (дисперсії) можна віднести до кожної із головних компонент.

Визначимо слідуючу функцію для коефіцієнту поглинання:

```
def ar(lambdas, proc):

# сортуємо власні значення у спадному порядку
    sorted_lambdas = np.sort(lambdas)[::-1]

# розраховуємо кумулятивну варіацію
    cumulative_variance = np.cumsum(sorted_lambdas) / np.sum(sorted_lambdas)

# знаходимо індекс, де кумулятивна варіація перетинає "proc"
    index_percent = np.argmax(cumulative_variance >= proc) +1

# виділяємо верхні власні значення, які описують встановлений відсоток даних
    selected_lambdas = sorted_lambdas[:index_percent]

# повертаємо коефіцієнт поглинання та кількість значень, що пояснюють обраний
відсоток варіації
    return np.sum(selected_lambdas)/np.sum(sorted_lambdas),
len(selected_lambdas)
```

Розглянемо, як варіюється коефіцієнт поглинання при різних значеннях поясненої варіації власними значеннями матриці кореляцій. На Рис. 8.5 представлено залежність коефіцієнта поглинання від різних значень поясненої варіації власними значеннями матриць кореляції вихідних прибутковостей та випадкових. Також представлено залежність кількості поясненої варіації від кількості власних значень, що пояснюють такий відсоток.

```
proc_variance = np.linspace(0, 0.99, 100)
ar_init = np.array([ar(lambdas, proc)[0] for proc in proc_variance])
ar_rand = np.array([ar(lambdas_rand, proc)[0] for proc in proc_variance])

lambdas_init_cnt = np.array([ar(lambdas, proc)[1] for proc in proc_variance])
```

```
lambdas_rand_cnt = np.array([ar(lambdas_rand, proc)[1] for proc in
proc_variance])

fig, ax = plt.subplots(1, 2, figsize=(8, 6))

ax[0].plot(proc_variance, ar_init, color='green', label=r'$AR_{init}$')
ax[0].plot(proc_variance, ar_rand, color='red', label=r'$AR_{rand}$')
ax[0].set_xlabel('Відсоток варіації\n(a)')
ax[0].set_ylabel('$AR$')
ax[0].legend()

ax[1].plot(proc_variance, lambdas_init_cnt, color='green',
label=r'$\#\lambda_{i}^{init}$')
ax[1].plot(proc_variance, lambdas_rand_cnt, color='red',
label=r'$\#\lambda_{i}^{rand}$')
ax[1].set_xlabel('Відсоток варіації\n(b)')
ax[1].set_ylabel(r'Кі-ть $\lambda_i$')
ax[1].legend()

fig.tight_layout()
plt.show();
```

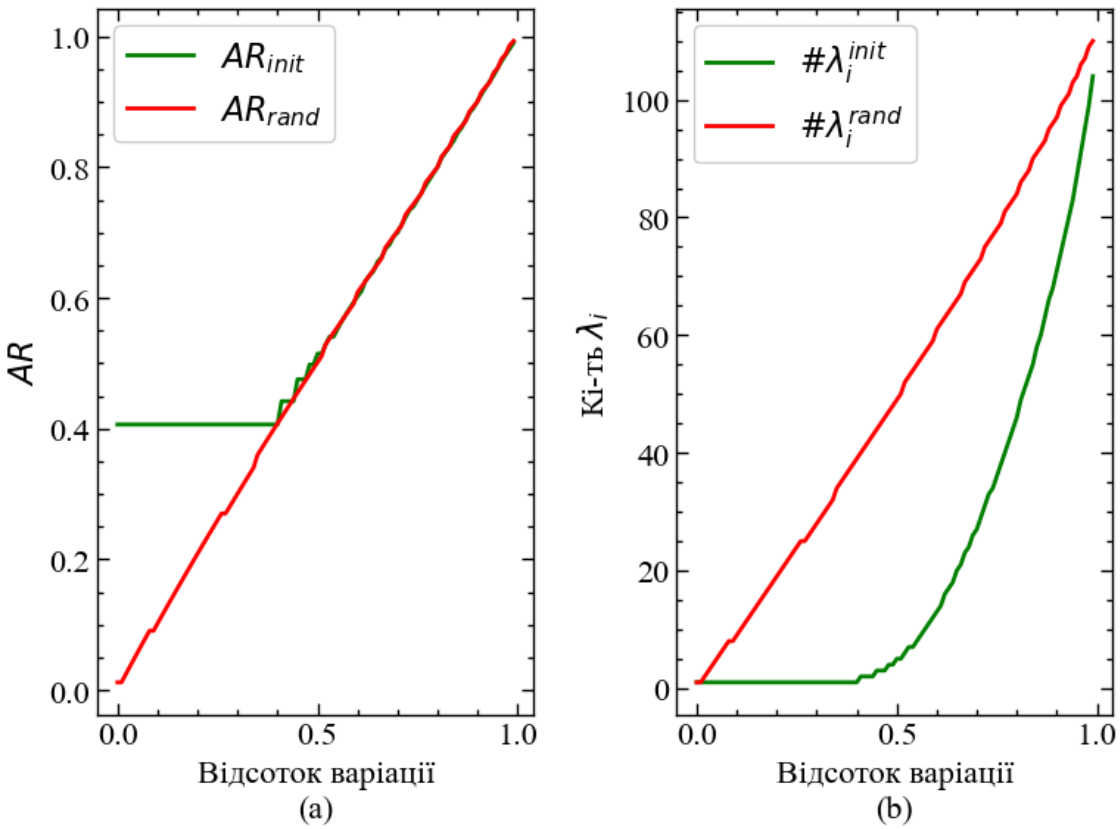


Рис. 8.5: Залежність коефіцієнта поглинання від різних значень поясненої варіації власними значеннями матриць кореляції вихідних прибутковостей та випадкових (a). Залежність кількості поясненої варіації від кількості власних значень, що пояснюють такий відсоток (b)

Як ми можемо бачити на Рис. 8.5, коефіцієнт поглинання для вихідної матриці кореляцій залишається сталим при перших 40% варіації. На рисунку справа видно, що лише одного власного значення достатньо для опису майже половини всього ринку. Для випадкової матриці кількість власних значень необхідних для опису системи зростає прямо пропоційно відсотку варіації. Отже, для нашої задачі при розрахунку коефіцієнту поглинання достатньо буде взяти до 40% варіації при врахуванні кількості необхідних власних значень у чисельнику (8.9).

8.2.5 Віконна процедура

Перш ніж розпочинати розрахунки визначимо функцію для побудови графіків. Вона буде виводити один із часових рядів досліджуваної бази акцій компаній та розраховані індикатори. Ми будемо виводити як двовимірні, так і тривимірні графіки.

```
def plot_2d(time_ser_index,
            time_ser_values,
            y1_values,
            time_ser_label,
            y1_label,
            x_label,
            file_name,
            clr="magenta"):

    fig, ax = plt.subplots(1, 1)

    ax2 = ax.twinx()

    ax2.spines.right.set_position(("axes", 1.03))

    p1, = ax.plot(time_ser_index,
                time_ser_values,
                "b-",
                label=fr"{time_ser_label}")
    p2, = ax2.plot(time_ser_index,
                y1_values, color=clr, label=y1_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=4, width=1.5)
    ax.tick_params(axis='x', rotation=45, **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();

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

length = data.shape[1]
num_bins = 50

mean_corr_init = []
mean_corr_rand = []

C_vals_init = []
C_vals_rand = []

C_hist_vals_init = []
C_hist_vals_rand = []

u_hist_vals_init = []
u_hist_vals_rand = []

lambdas_vals_init = []
lambdas_vals_rand = []

lambdas_vals_max_init = []
lambdas_vals_max_rand = []

lambdas_hist_vals_init = []
lambdas_hist_vals_rand = []

ipr_vals_init = []
ipr_vals_rand = []

absorb_vals_init = []
absorb_vals_rand = []

for i in tqdm(range(0, length-window, tstep)):

# відбираємо фрагменти
    fragm = data.iloc[:, i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)
    fragm = fragm[~np.isnan(fragm).any(axis=1)]
```

```
# генеруємо випадковий фрагмент
    fragm_random = np.random.normal(size=(fragm.shape[0], fragm.shape[1]))

# розраховуємо крос-кореляції
    C_fragm, ccoef_flat_fragm = calc_cross_corr(fragm)
    C_fragm_random, ccoef_flat_fragm_random = calc_cross_corr(fragm_random)

# розраховуємо гістограму крос-кореляцій
    C_fragm_hist = np.histogram(ccoef_flat_fragm, bins=num_bins,
density=True)[0]
    C_fragm_hist_rand = np.histogram(ccoef_flat_fragm_random, bins=num_bins,
density=True)[0]

# розрахунок середнього значення коефіцієнта кореляції
    mean_correlation = np.mean(ccoef_flat_fragm)
    mean_correlation_random = np.mean(ccoef_flat_fragm_random)

# розрахунок власних значень та векторів
    lambdas_fragm, u_fragm = calc_lambd_eig(C_fragm)
    lambdas_fragm_random, u_fragm_random = calc_lambd_eig(C_fragm_random)

# розраховуємо гістограму власних значень
    lambdas_fragm_hist = np.histogram(lambdas_fragm, bins=num_bins,
density=True)[0]
    u_fragm_hist_init = np.histogram(u_fragm, bins=num_bins, density=True)[0]
    lambdas_fragm_hist_rand = np.histogram(lambdas_fragm_random, bins=num_bins,
density=True)[0]
    u_fragm_hist_rand = np.histogram(u_fragm_random, bins=num_bins,
density=True)[0]

# отримання максимального власного значення
    lambdas_fragm_max = lambdas_fragm.max()
    lambdas_fragm_max_random = lambdas_fragm_random.max()

# отримання оберненого відношення участі
    ipr_fragm = calc_ipr(u_fragm)
    ipr_random = calc_ipr(u_fragm_random)

# розрахунок коефіцієнта поглинання
    absorb_init = ar(lambdas_fragm, 0.05)[0]
    absorb_rand = ar(lambdas_fragm_random, 0.05)[0]

# додаємо індикатори до масивів
    mean_corr_init.append(mean_correlation)
    mean_corr_rand.append(mean_correlation_random)

    C_vals_init.append(ccoef_flat_fragm)
    C_vals_rand.append(ccoef_flat_fragm_random)

    C_hist_vals_init.append(C_fragm_hist)
    C_hist_vals_rand.append(C_fragm_hist_rand)
    lambdas_hist_vals_init.append(lambdas_fragm_hist)
    lambdas_hist_vals_rand.append(lambdas_fragm_hist_rand)
    u_hist_vals_init.append(u_fragm_hist_init)
```

```
    u_hist_vals_rand.append(u_fragm_hist_rand)

    lambdas_vals_init.append(lambdas_fragm)
    lambdas_vals_rand.append(lambdas_fragm_random)

    lambdas_vals_max_init.append(lambdas_fragm_max)
    lambdas_vals_max_rand.append(lambdas_fragm_max_random)

    ipr_vals_init.append(ipr_fragm)
    ipr_vals_rand.append(ipr_random)

    absorb_vals_init.append(absorb_init)
    absorb_vals_rand.append(absorb_rand)

ind_names = ['avg_corr_init', 'lambda_max', 'ar']

indicators = [mean_corr_init, lambdas_vals_max_init, absorb_vals_init]

measure_labels = [r'$\langle C \rangle_{init}$', r'$\lambda_{max}$', r'$AR$']

for i in range(len(ind_names)):
    name =
f"RMT_{ind_names[i]}_symbol={ylabel}_wind={window}_step={tstep}_seriestype={ret_
type}"
    np.savetxt(name +".txt", indicators[i])
```

8.2.5.1 Середній коефіцієнт крос-кореляцій

```
file_name =
f"avg_corr_symbol={ylabel}_wind={window}_step={tstep}_seriestype={ret_type}"

plot_2d(data_init.loc[:, ylabel].index[window:length:tstep],
        data_init.loc[:, ylabel].values[window:length:tstep],
        mean_corr_init,
        ylabel,
        measure_labels[0],
        xlabel,
        file_name,
        clr="magenta")
```

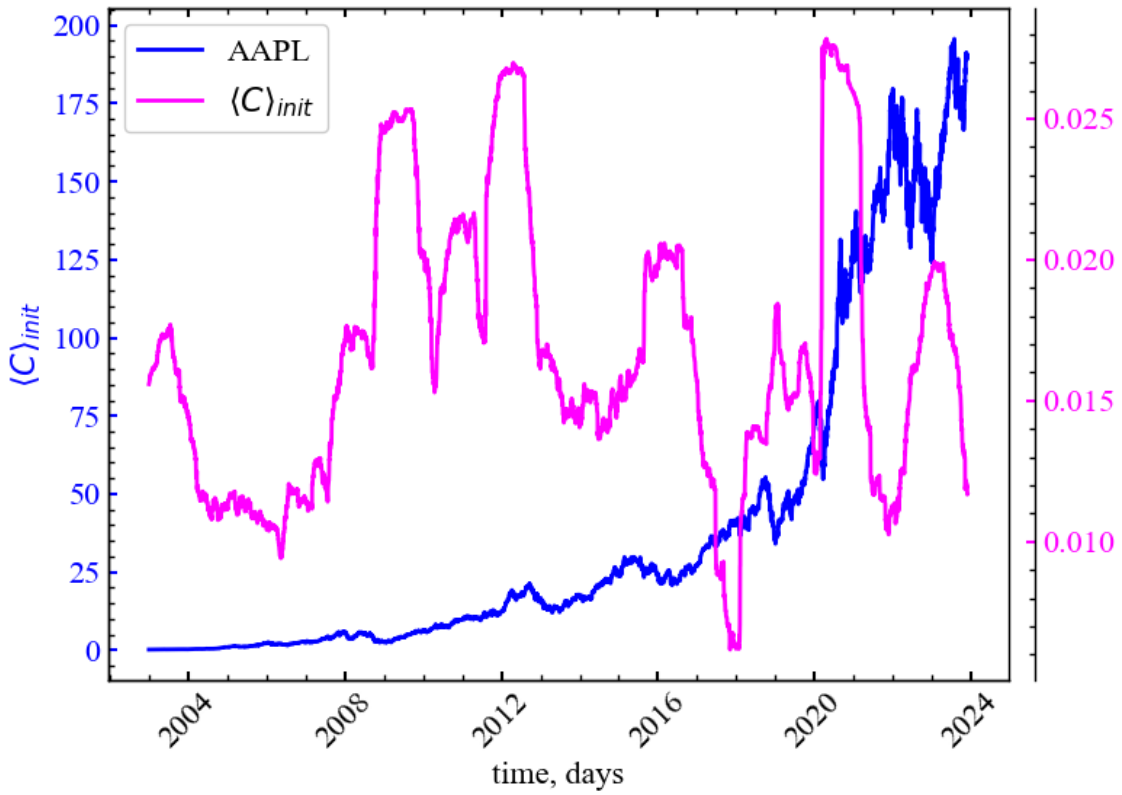


Рис. 8.6: Динаміка індексу цін акцій компанії Apple та середнього показника крос-кореляцій прибутковостей досліджуваних акцій

На Рис. 8.6 видно, що середній ринковий ступінь крос-кореляцій зростає у (перед-)кризові періоди, що вказує на зростання самоорганізованості ринку. Усі індекси синхронізовано починають “відповідати” на зовнішні чинники, які змушують трейдерів колективно все відкуповувати або розпродавати.

На Рис. 8.7 представлений той самий показник, але у тривимірному просторі.

```
Y = np.linspace(-1, 1, num_bins)

X = np.arange(window, length, tstep)
X = np.expand_dims(X, axis=1)
X = np.repeat(a=X, repeats=Y.shape[0], axis=1)

Z = np.array(C_hist_vals_init)

fig, ax = plt.subplots(subplot_kw={"projection": "3d"}, figsize=(8, 6))

surf = ax.plot_surface(X, Y, Z, cmap='hot', rstride=2, cstride=2, linewidth=0)

ax.set_xlabel(xlabel, fontsize=16, labelpad=15)
ax.set_ylabel(r"$C_{ij}$", fontsize=16, labelpad=15)
ax.set_zlabel(r"$P(C)$", fontsize=16, labelpad=15)
ax.tick_params(axis='both', which='major', labelsize=16, pad=5)

ax.view_init(60, 140)
```

```
fig.tight_layout()

plt.savefig(f"RMT_3D_P(C)_wind={window}_step={tstep}_seriestype={ret_type}.jpg",
bbox_inches="tight")

plt.show();
```

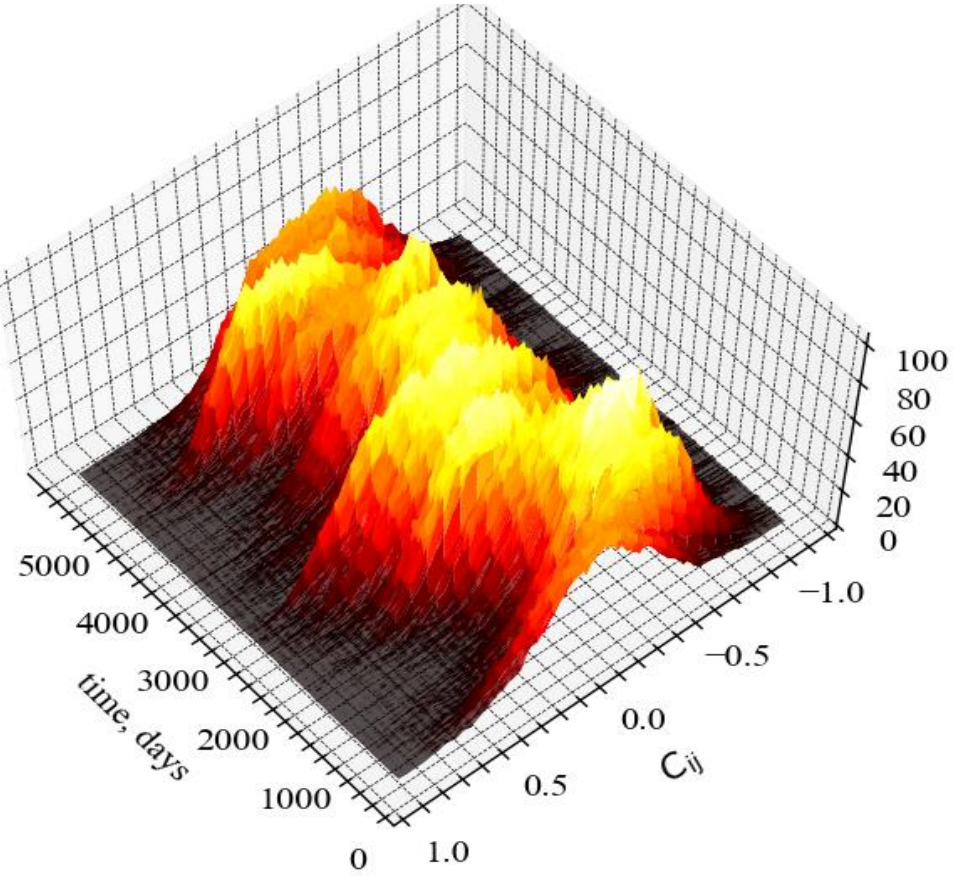


Рис. 8.7: Віконна функція щільності розподілу коефіцієнтів кореляції вихідної матриці фінансових активів

8.2.5.2 Максимальне значення λ

На наступному рисунку (Рис. 8.8) представлено порівняльну динаміку індексу цін акцій компанії Apple та показника λ_{max}.

```
file_name =
f"lambda_max_symbol={ylabel}_wind={window}_step={tstep}_seriestype={ret_type}"

plot_2d(data_init.loc[:, ylabel].index[window:length:tstep],
        data_init.loc[:, ylabel].values[window:length:tstep],
        lambdas_vals_max_init,
        ylabel,
        measure_labels[1],
        xlabel,
        file_name,
        clr="red")
```

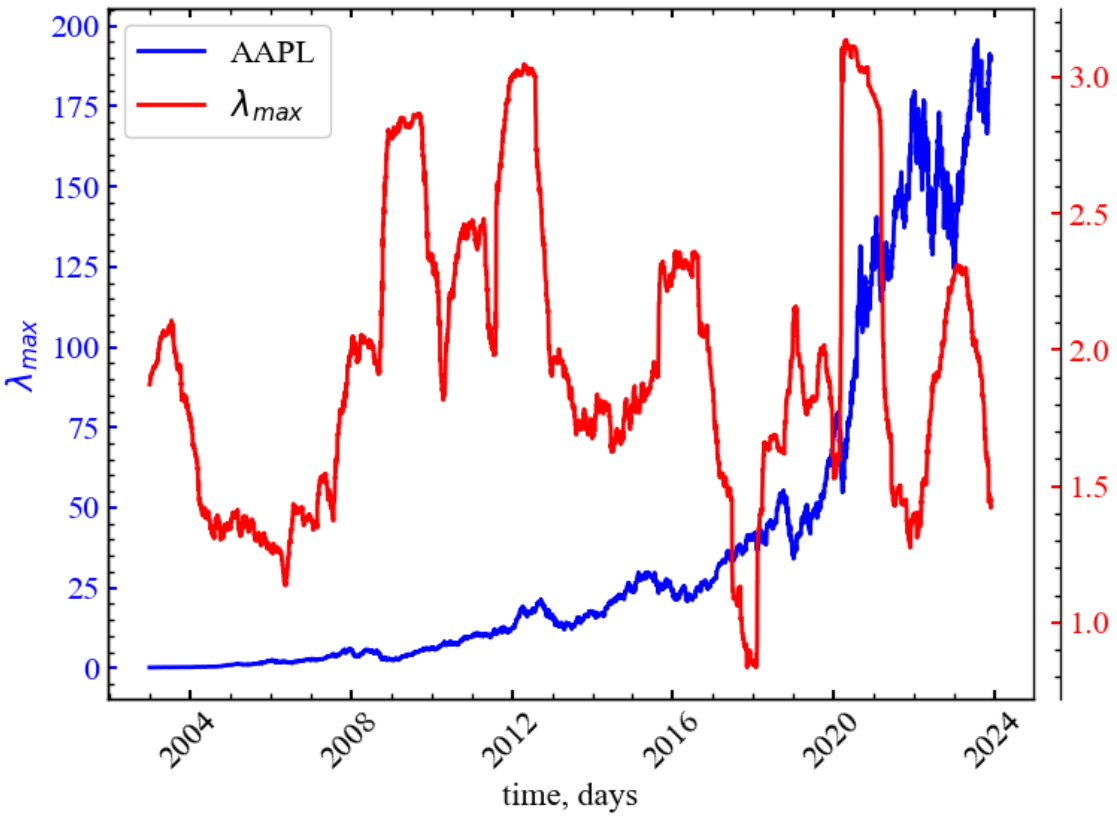


Рис. 8.8: Динаміка індексу цін акцій компанії Apple та показника λ_{max}

Рис. 8.8 показує, що λ_{max} у схожий спосіб із $\langle C \rangle_{init}$. Із динаміки даного показника можна зробити висновок, що серед усіх індексів найбільш впливовим є індекс, якому характерно λ_{max}. Усі інші індекси, власне значення яких знаходиться в межах теоретичного передбачення ТВМ, вносять лише шумову інформацію в загальну динаміку ринку і реагують на всі інші збурення на фондовому ринку, слідуючи закономірностям найбільш капіталізованих індексів.

8.2.5.3 Обернене відношення участі

На Рис. 8.9 представлено тривимірну віконну динаміка показника оберненого відношення участі

```
def log_tick_formatter(val, pos=None):
    return f"$10^{{{int(val)}}}$"

Y = np.array(lambdas_vals_init)

X = np.arange(window, length, tstep)
X = np.expand_dims(X, axis=1)
X = np.repeat(a=X, repeats=Y.shape[1], axis=1)

Z = np.array(ipr_vals_init)
```

```
fig, ax = plt.subplots(subplot_kw={"projection": "3d"}, figsize=(8, 6))

surf = ax.plot_surface(X, np.log(Y), Z, cmap='magma', rstride=2, cstride=2,
linewidth=0.5)

ax.set_xlabel(xlabel, fontsize=16, labelpad=15)
ax.set_ylabel(r"$\lambda_i$", fontsize=16, labelpad=15)
ax.set_zlabel(r"$IPR$", fontsize=16, labelpad=15)
ax.tick_params(axis='both', which='major', labelsize=16, pad=5)

ax.yaxis.set_major_formatter(mticker.FuncFormatter(log_tick_formatter))
ax.yaxis.set_major_locator(mticker.MaxNLocator(integer=True))

fig.tight_layout()

plt.savefig(f"RMT_3D_IPR_wind={window}_step={tstep}_seriestype={ret_type}.jpg",
bbox_inches="tight")

plt.show();
```

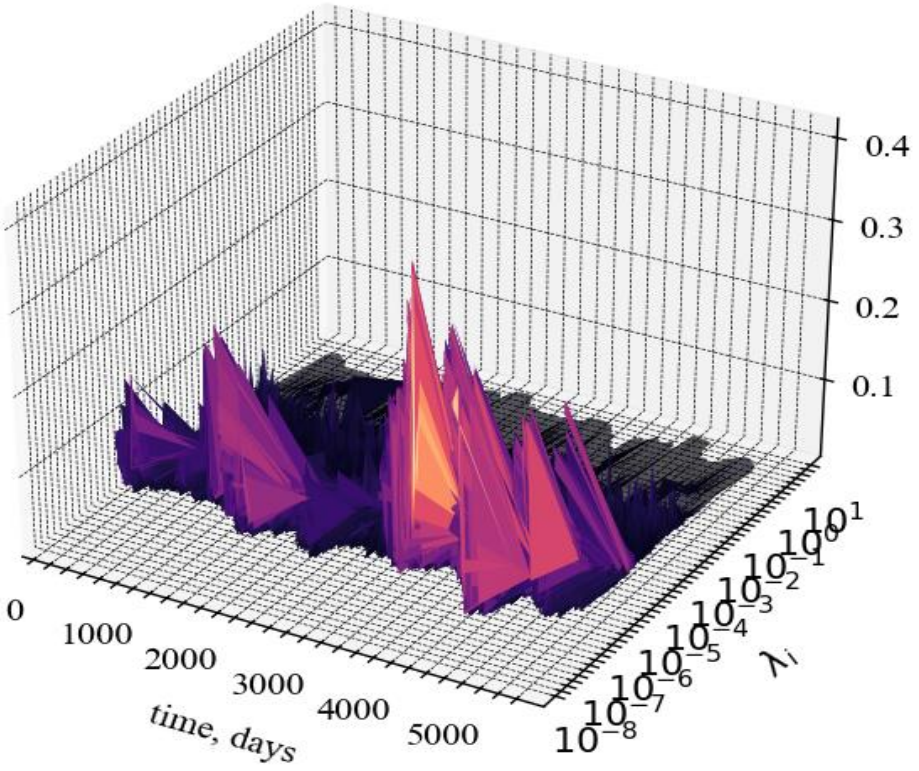


Рис. 8.9: Тривимірна віконна динаміка показника оберненого відношення участі

ОВУ на Рис. 8.9 характеризується значним зростанням у кризові періоди, у той час як для періодів релаксації цей показник залишається на рівні нуля.

8.2.5.4 Коефіцієнт поглинання

На Рис. 8.10 представлено порівняльну динаміку індексу цін акцій компаній Apple та коефіцієнту поглинання.

```
file_name =
f"ar_symbol={ylabel}_wind={window}_step={tstep}_seriestype={ret_type}"

plot_2d(data_init.loc[:, ylabel].index[window:length:tstep],
        data_init.loc[:, ylabel].values[window:length:tstep],
        absorb_vals_init,
        ylabel,
        measure_labels[2],
        xlabel,
        file_name,
        clr="black")
```

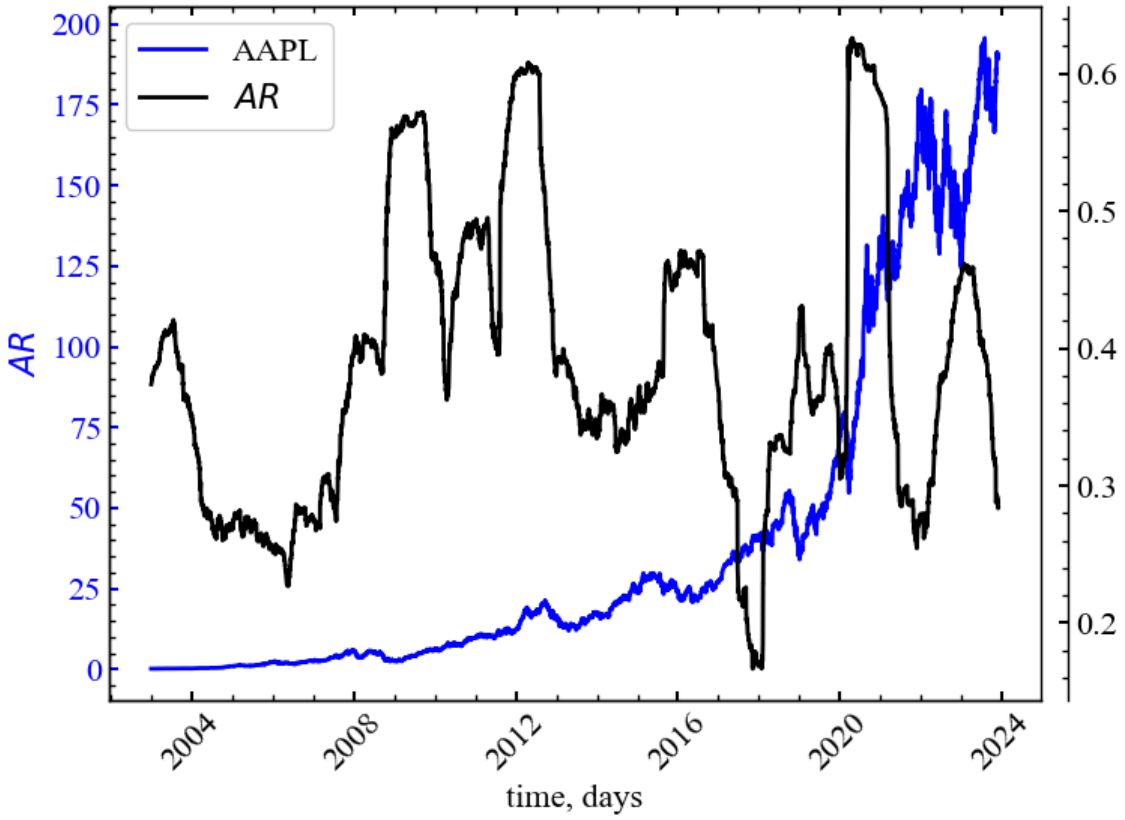


Рис. 8.10: Динаміка індексу цін акцій компаній Apple та коефіцієнту поглинання

На Рис. 8.10 видно, що *AR* зростає під час крахів 2008, 2015-2016, 2021 і 2023 року. Це говорить про те, що в кризові періоди зростає активність одного або декілька індексів, які стають рушієм для усього фондового ринку.

8.3 Висновок

Таким чином, при наявності сукупності часових рядів, що є даними діяльності економічних об’єктів однієї області, можна провести дослідження

стосовно структури вказаної області та взаємодії об’єктів всередині неї. Дослідження проводяться на основі теорії випадкових матриць, що дозволяє отримувати інформацію шляхом аналізу матриці крос-кореляцій, побудованої для сукупної бази економічних об’єктів.

8.4 Завдання для самостійної роботи

Оберіть певний фондовий індекс і проведіть дослідження крахових подій для ринку, що він представляє за допомогою теорії випадкових матриць.

8.5 Контрольні запитання

1. Поясніть основну ідею теорії випадкових матриць
2. Про що свідчить відмінність кореляційних і спектральних властивостей матриці даних і випадкової?
3. Дослідіть, як змінюється розподіл власних значень у випадках:
   - $\lambda_- < \lambda < \lambda_+$;
   - $\lambda > \lambda_+$;
   - $\lambda < \lambda_-$.
4. Порівняйте кольорову карту поля взаємних кореляцій випадкової матриці і заданої. Зробіть висновки

9. Лабораторна робота № 9

Тема. Показники Ляпунова та стійкість складних систем

Мета. Навчитися розпізавати та передчасно ідентифікувати катастрофічні події, використовуючи показники Ляпунова

9.1 Теоретичні відомості

Здавалося б, випадкові коливання у складних системах часто демонструють різний рівень складності та хаотичності. В умовах обмеженості даних стає важко визначити межі їх передбачуваності. Аналіз таких систем, процесів, що визначають їх динаміку, теорія хаосу розглядалися в різних галузях, таких як економіка, фінанси, фізика та ін. Що стосується аналізу, наприклад, динаміки Біткоїна, то знання про його абсолютно випадкові і, водночас, детерміновані процеси потенційно можуть пояснити флуктуації часових рядів різної природи. Протягом багатьох років теорія хаосу надавала підходи до вивчення деяких цікавих властивостей часових рядів. Найбільш поширеними є: кореляційна розмірність, BDS тест, ентропія Колмогорова, показники Ляпунова тощо.

Продемонструємо, яким чином показники Ляпунова дають можливість дослідити режими хаотичної та детермінованої поведінки.

9.1.1 Показники Ляпунова

Еволюція системи демонструє **чутливість до початкових умов** (sensitivity to initial conditions). Це означає, що спочатку близькі траєкторії, які розвиваються, можуть швидко відхилятися одна від одної і мати абсолютно різні результати. Відповідно, при малих невизначеностях, які надзвичайно швидко посилюються, довгострокові прогнози виявляються неможливими. З іншого боку, в системі з точками тяжіння або стабільними точками відстань між ними асимптотично зменшується з часом або з кількістю точок, які мають тенденцію до зближення [141].

Щоб представити ідею більш точно, розглянемо дві послідовні траєкторії — $x(t)$ та наближчого сусіда цієї траєкторії з невеликим зміщенням, $x(t) + \delta(t)$, де $\delta(t)$ представляє собою мале відхилення в часі t, як показано на Рис. 9.1.

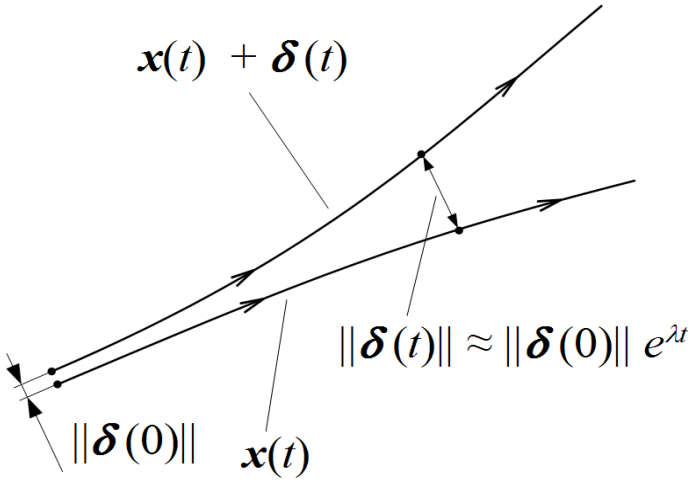


Рис. 9.1: Розбіжність двох початково близьких траєкторій [142]

Коли динаміка двох початково близьких траєкторій порушується певною подією відстань між ними може зростати експоненційно [143]:

$$\|\delta(t)\| \approx \|\delta(0)\| \exp(\lambda t), \tag{9.1}$$

де λ позначає **показник Ляпунова** (ПЛ); $\delta(t)$ — відстань між точкою що розглядається та її наближчим сусідом після t ітерацій; $\delta(0)$ — початкова відстань між точкою, що розглядається та її найближчим сусідом у початковий момент часу ($t = 0$).

ПЛ є мірою швидкості експоненціальної розбіжності близьких один до одного траєкторій у фазовому просторі динамічної системи. Іншими словами, ПЛ показує, наскільки швидко зближуються або розходяться траєкторії, які починаються близько одна від одної, вимірюючи ступінь хаосу в системі.

У тих випадках, коли наша система n-вимірна, ми маємо стільки ж ПЛ. Для їх визначення розглянемо еволюцію нескінченно малої сфери, що зазнала збурень за різними осями. Визначивши величину збурення по вісі i як $\delta_i(t)$, отримаємо n **показників Ляпунова**, що мають вид

$$\|\delta_i(t)\| \approx \|\delta_i(0)\| \exp(\lambda_i t), \text{ для } i = 1, \dots, n. \tag{9.2}$$

Для визначення того, чи є рух періодичним або хаотичним, особливо для великих t, рекомендується розглядати внесок системи в **найбільший показник Ляпунова** (НПЛ), оскільки діаметр n-розмірного еліпсоїда починає залежати від нього [142]. Саме НПЛ використовується для кількісної оцінки передбачуваності систем, оскільки експоненціальна розбіжність означає, що в

системі, де початкове збурення було нескінченно малим, починаються втрати передбачуванності. Однак слід зазначити, що інші показники також містять важливу інформацію щодо стійкості системи, в тому числі про напрямки збіжності та розбіжності траєкторій [144].

Існування принаймні одного позитивного ПЛ зазвичай розглядається як сильний індикатор хаосу. Позитивний ПЛ означає, що початково близькі траєкторії у фазовому просторі, чутливі до початкових умов і розходяться експоненціально швидко. Негативний ПЛ відповідає випадкам, коли траєкторії залишаються близькими одна до одної, але це не обов'язково означає стабільність, і ми повинні дослідити нашу систему більш детально. Нульові або дуже близькі до нуля показники вказують на те, що збурення, практично не впливають на еволюцію траєкторій динамічної системи.

У зв'язку з великою зацікавленістю в ПЛ, з'являється все більше інструментів для розрахунку. На жаль, досі не отримано загальноприйнятого та універсального методу оцінки всього спектру показників Ляпунова за значеннями часового ряду. Одні з найбільш поширених і популярних алгоритмів були застосовані Вольфом та ін. [145], Сано і Савадою [146], а пізніше вдосконалені Екманом [147], Розенштейном [11], Парліцом [148], Бальцержаком тощо [149].

9.1.1.1 Метод Екмана

По-перше, згідно з підходом Екмана та ін. [147], ми повинні реконструювати динаміку атрактора з часового ряду $\{x(i) \mid i = 1, \dots N\}$ з розмірністю вкладень d_E, і після цього побудувати d_E-вимірну орбіту, що представляє часову еволюцію

$$\vec{X}(i) = \left[x(i), x(i+1), \dots, x\left(i + (d_E - 1)\right)\right], \text{ для } i = 1, \dots, N - d_E + 1.$$

Далі, ми маємо визначити найближчі до $\vec{X}(i)$ траєкторії:

$$\left\|\vec{X}(i) - \vec{X}(j)\right\| = \max_{0 \le \alpha \le d_E - 1} |x(i+\alpha) - x(j+\alpha)|. \tag{9.3}$$

Сортуємо $x(i)$ так, щоб $x\left(\Pi(1)\right) \le x\left(\Pi(2)\right) \le \dots \le x\left(\Pi(N)\right)$ і зберігаємо перестановку Π та її зворотню версію $\Pi^{-}1$. Далі намагаємось знайти сусідів $x(i)$, переглядаючи $k = \Pi^{-1}(i)$ і скануємо $x\left(\Pi(s)\right)$ при $s = k+1, k+2, \dots$ і $k-1, k-2, \dots$ до тих пір, до поки не виконається умова $x\left(\Pi(s)\right) - x(i) > r$. Для вибраної розмірності вкладення $d_E > 1$ вибираємо вибираємо значення s за умови

$$|x(П(s)+\alpha)-x(j+\alpha)| \le r, \text{ для } \alpha = 0,1,\dots,d_E-1.$$

Після реконструкції систем до розмірності d_E потрібно визначити матрицю M_i розмірності $d_E \times d_E$, яка описуватиме часову еволюцію векторів, із оточення траєкторії $\vec{X}(i)$, і те, як вони відображаються на стан $\vec{X}(i+1)$. Матриця M_i отримується шляхом пошуку сусідів

$$M_i\left(\vec{X}(i)-\vec{X}(j)\right) \approx \vec{X}(i+1)-\vec{X}(j+1). \quad (9.4)$$

Вектори $\vec{X}(i)-\vec{X}(j)$ можуть і не покривати $\mathbb{R}^{d_E}$. У цьому випадку така невизначеність може призвести до хибних показників, які можуть зіпсувати аналіз. Для подолання таких перешкод проекція траєкторій визначається на підпросторі розмірності $d_M \le d_E$. Таким чином, простір, на якому відбувається динаміка, відповідає локальній розмірності d_M, а d_E має бути дещо більшим за d_M, щоб уникнути наявності хибних сусідів [62,150]. Звідси випливає, що траєкторія $\vec{X}(i)$ асоціюється з d_M-вимірним вектором

$$\vec{X}(i) = [x(i), x(i+\tau), \dots, x(i+(d_M-1)\tau)] = [x(i), x(i+\tau), \dots, x(i+d_E-1)], \quad (9.5)$$

де $\tau = (d_E-1)/(d_M-1)$. Коли $\tau > 1$, умова (9.4) замінюється наступним виразом:

$$M_i\left(\vec{X}(i)-\vec{X}(j)\right) \approx \vec{X}(i+\tau)-\vec{X}(j+\tau). \quad (9.6)$$

Матриця M_i визначається методом найменших квадратів. Останнім кроком є QR декомпозиція для знаходження ортогональних матриць Q_i і верхніх трикутних матриць R_i при яких

$$M_{1+i\tau}Q_i = Q_{i+1}R_{i+1}, \text{ для } i = 0,1,2,\dots .$$

Як було запропоновано Екманом [147,151], знаючи K кількість точок на атракторі, діагональні власні значення матриці R_i та крок дискретизації Δt, можна визначити наступне рівняння для знаходження k-го ПЛ:

$$\lambda_k = \frac{1}{\Delta t}\frac{1}{\tau}\frac{1}{K}\sum_{i=0}^{K-1}\ln(R_i)_{kk}.$$

9.1.1.2 Метод Розенштейна

Алгоритм Розенштейна [11] використовує метод реконструкції вкладень із часовою затримкою, який передає найважливіші особливості багатовимірного атрактора в один одновимірний часовий ряд деякого скінченного розміру N. Для часового ряду кожен вектор $\vec{X}(i)$ буде представлений подібно до вектора (9.5) із розмірністю вкладень d_E і часовою затримкою τ. Потім на відновленій траєкторії ми ініціалізуємо пошук у просторі станів найближчого сусіда $\vec{X}(j)$ для траєкторії $\vec{X}(i)$:

$$\delta_i(0) = \min_{\vec{X}(i)} \left\| \vec{X}(i) - \vec{X}(j) \right\|, \text{ для } |i-j| > \text{середній період},$$

де $\|\cdot\|$ — це Евклідова норма, $\vec{X}(j)$ — найближча сусідня траєкторія, $\vec{X}(i)$ — розглядувана траєкторія.

З (9.1) ми вже знаємо, що відстань між станами $\vec{X}(i)$ та $\vec{X}(j)$ зростає з часом відповідно до степеневого закону, де λ є хорошим наближенням СПЛ. Для подальших оцінок розглянемо логарифм відстані на траєкторії $\ln \delta_i(k) \approx \lambda(k \cdot \Delta t) + \ln c_i$, де $\delta_i(k)$ — відстань між i-ою парою найближчих сусідів, визначених рівнянням (9.6) через k часових кроків, c_i — початкова відстань між ними, а Δt — часовий інтервал між вимірюваннями (період дискретизації часового ряду).

Подальший результат цього алгоритму представляє функцію від часу

$$y(k, \Delta t) = \frac{1}{M \Delta t} \sum_{i=1}^{M} \ln \delta_i(k),$$

де $M = N - (d_E - 1)\tau$ — розмір реконструйованого часового ряду, а $\delta_i(k)$ — i-та лінія, нахил котрої приблизно рівний СПЛ. Тоді пропонується обчислювати СПЛ як кут нахилу найбільш лінійної ділянки. Знаходження такої ділянки виявляється нетривіальною задачею. Незважаючи на цю проблему, метод Розенштейна є простим для реалізації та обчислення.

9.2 Хід роботи

Розглянемо, як можна використовувати зазначені підходи для розрахунку відповідних хаос-динамічних індикаторів. Спочатку імпортуємо необхідні бібліотеки.

```
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

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                   # розмір фонтів рисунку
'lines.linewidth': 2,                # товщина ліній
'axes.titlesize': 'small',           # розмір титулки над рисунком
'axes.labelsize': size,              # розмір підписів по осям
'legend.fontsize': size,             # розмір легенди
'xtick.labelsize': size,             # розмір розмітки по осі Ox
'ytick.labelsize': size,             # розмір розмітки по осі Ox
"font.family": "Serif",              # сімейство стилів підписів
"font.serif": ["Times New Roman"],   # стиль підпису
'savefig.dpi': 300,                  # якість збережених зображень
'axes.grid': False                   # побудова сітки на самому рисунку
}

plt.rcParams.update(params)          # оновлення стилю згідно налаштувань
```

Розглянемо значення фондового індексу Доу Джонса за весь період, що представляє Yahoo! Finance. В якості кінцевої дати зазначимо 1 грудня 2023 року.

```
symbol = '^DJI'                       # Символ індексу
end = '2023-12-01'                    # кінцева дата
data = yf.download(symbol, end=end)   # вивантажуємо дані
time_ser = data['Adj Close'].copy()   # зберігаємо саме ціни закриття

xlabel = 'time, days'                 # підпис по вісі Ox
ylabel = symbol                       # підпис по вісі Oy
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того, з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'           # підпис по вісі Ox
ylabel = symbol                     # підпис по вісі Oy
```

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 Ox
ax.set_ylabel(ylabel)                           # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                         # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                    # Зберігаємо графік
plt.show();                                     # Виводимо графік
```

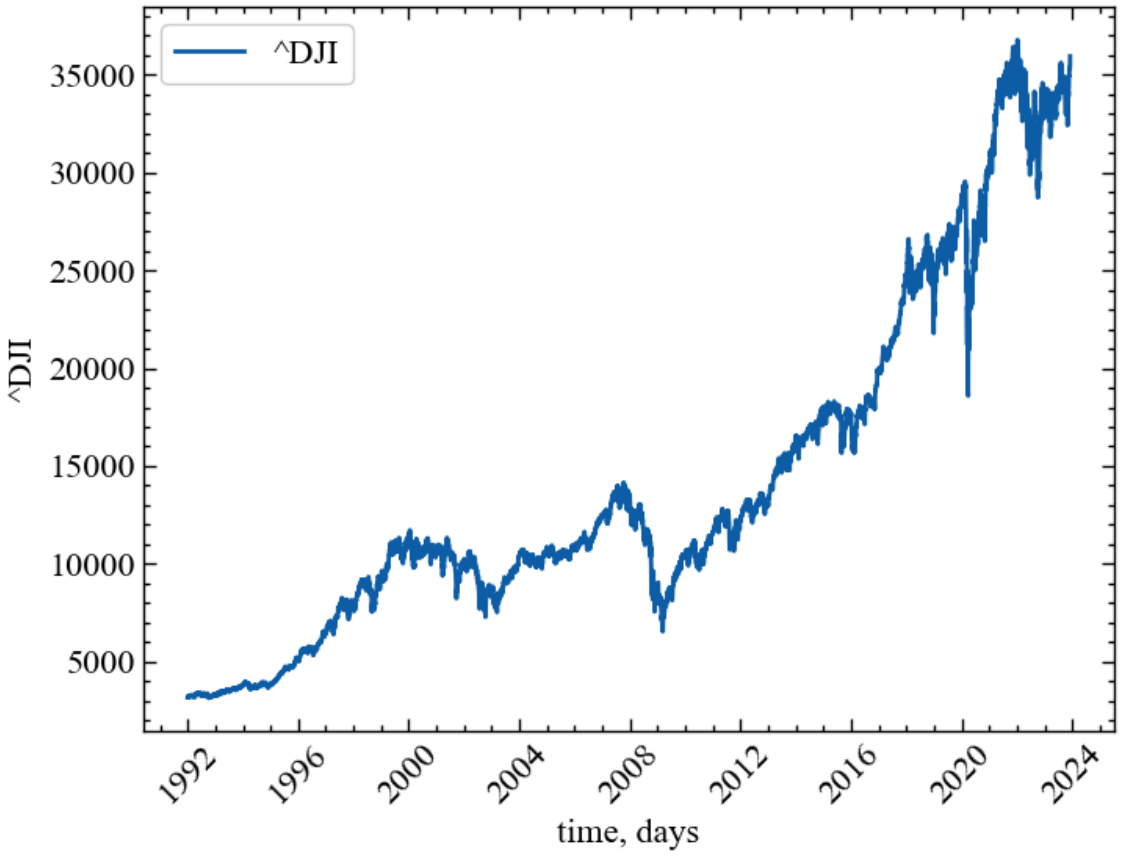


Рис. 9.2: Динаміка щоденних значень фондового індексу Доу Джонса

Визначимо функцію `transformation()` для виконання перетворення ряду до прибутковостей або стандартизованих значень:

```
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
```

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=4, width=1.5)
    ax.tick_params(rotation=45, 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();
```

9.2.1 Обчислення показників Ляпунова із використанням віконної процедури

Для подальших розрахунків використовуватимемо бібліотеку `neurokit2`. Ключовою функцією для отримання відповідних показників є `complexity_lyapunov()`. Вона надає доступ до розрахунків згідно з наступними алгоритмами:

- **Розенштейна та ін. (1993)**;
- **Маковскі** — це спеціальна модифікація алгоритму Розенштейна, що використовує процедуру k-вимірного дерева для більш ефективного обчислення найближчих сусідів. Крім того, СПЛ обчислюється як нахил до точки зміни швидкості розбіжності (точки, де вона вирівнюється), що робить його більш стійким до параметра довжини траєкторії;
- **Екман та ін. (1986)**.

Розглянемо її синтаксис більш детально:

`complexity_lyapunov(signal, delay=1, dimension=2, method='rosenstein1993', separation='auto', **kwargs)`

Параметри:

- **signal** (*Union[list, np.array, pd.Series]*) — сигнал;
- **delay** (*int*) — часова затримка (часто позначається τ іноді називають запізненням);
- **dimension** (*int*) — розмірність вкладень (d_E, іноді позначається як d або порядок). Якщо метод має значення `"eckmann1986"`, рекомендується використовувати більші значення розмірності;
- **method** (*str*) — метод, який визначає алгоритм обчислення ПЛ. Може бути `"rosenstein1993"`, `"makowski"` або `"eckmann1986"`;

- **len_trajectory** (*int*) — застосовується, якщо метод `"rosenstein1993"`. Кількість точок даних, в яких простежуються сусідні траєкторії;
- **matrix_dim** (*int*) — застосовується, якщо метод `"eckmann1986"`. Відповідає кількості ПЛ, які потрібно повернути;
- **min_neighbors** (*int*, *str*) — застосовується, якщо метод `"eckmann1986"`. Мінімальна кількість сусідів. Якщо `"default"`, використовується `min(2 * matrix_dim, matrix_dim + 4)`;
- **kwargs** (необов'язково) — інші аргументи, які передаються до `signal_psd()` для обчислення мінімального часового розділення двох сусідів.

Повертає:

- **lle** (*float*) — оцінка СПЛ, якщо метод `"rosenstein1993"`, і масив ПЛ, якщо `"eckmann1986"`;
- **info** (*dict*) — словник, що містить додаткову інформацію щодо параметрів, які використовуються для обчислення СПЛ.

Перед розрахунками виконаємо оновлення бібліотеки `neurokit2`:

```
!pip install --upgrade neurokit2
```

9.2.1.1 Обчислення старшого показника Ляпунова на основі методу Розенштейна

Спочатку виконаємо розрахунки для **всього** ряду індексу Доу Джонса:

```
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

time_ser_ret = transformation(signal, ret_type)
```

Далі визначимо наступні параметри:

```
d_E = 3                    # розмірність вкладень
tau = 10                   # часові затримка
approach_lyap = "makowski" # метод для розрахунку старшого показника
max_len = "auto" # встановлюємо максимальну довжину траєкторії у 10 разів більшу
за затримку
sep = "auto" # оцінка середнього періоду як величину, обернену до середньої
частоти спектра потужності
```

і візуалізуємо результат:

```
lle, _ = nk.complexity_lyapunov(signal=time_ser_ret,
                                method=approach_lyap,
                                dimension=d_E,
```

```
                                delay=tau,
                                max_length=max_len,
                                separation=sep,
                                show=True)
```

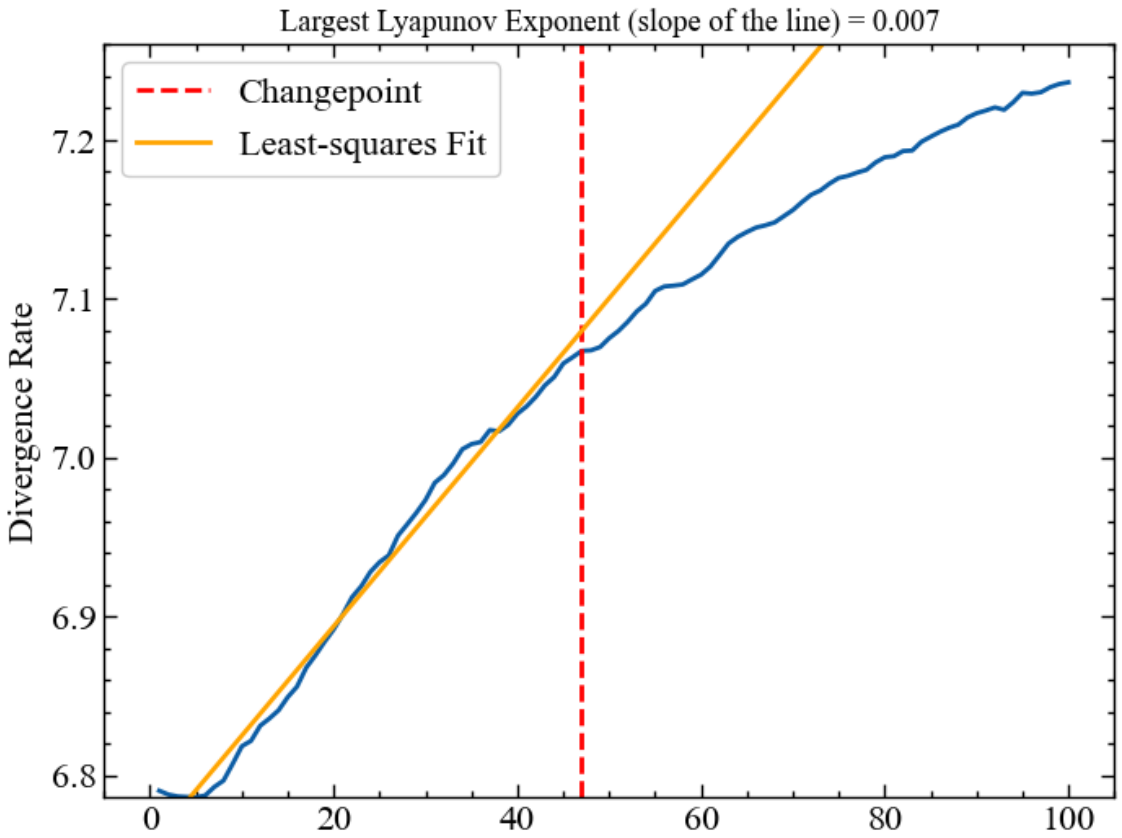


Рис. 9.3: Діаграма розбіжності траєкторій реконструйованого фазового простору індексу Доу Джонса, що представляє розрахований СПЛ

На Рис. 9.3 показано типовий графік (суцільна крива) залежності середньої розбіжності траєкторій від часу Δt; помаранчева лінія має нахил, що дорівнює теоретичному значенню λ_{max}. Коротка синя ділянка до переходу через червону пунктирну лінію використовується для вилучення найбільшого показника Ляпунова. Як ми можемо бачити, крива змінюється при більших часових періодах, оскільки система обмежена у фазовому просторі і середня дивергенція не може перевищувати “довжину” атрактора. Отриманий показник Ляпунова вказує на те, що індекс Доу Джонса знаходиться на межі між хаосом та стабільність, тобто індекс дивергенції динаміки ряду врівноважується конвергенцією.

Як ми вже мали змогу переконатись, складні системи мінливі і система з плином часу може проявляти як конвергенцію чи дивергенцію, так і повну незмінність у часі.

Далі розглянемо динаміку досліджуваної системи з часом у рамках процедури **ковзного вікна**. Визначимо наступні параметри:

```
window = 500 # ширина вікна
tstep = 1    # часовий крок вікна
length = len(time_ser)  # довжина самого ряду
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_E = 3 # розмірність вкладень
tau = 1 # часові затримка
approach_lyap = "makowski" # метод для розрахунку старшого показника:
rosenstein1993, makowski
max_len = "auto" # встановлюємо максимальну довжину траєкторії у 10 разів більшу
за затримку: auto
sep = "auto" # оцінка середнього періоду як величину, обернену до середньої
частоти спектра потужності

LLE = []                # масив для збереження СПЛ
```

Тепер можна приступати 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

    lle, _ = nk.complexity_lyapunov(signal=fragm,
                                    method=approach_lyap,
                                    dimension=d_E,
                                    delay=tau,
                                    max_length=max_len,
                                    separation=sep,
                                    show=False)

    LLE.append(lle)
```

Зберігаємо отримані результати в текстовому файлі:

```
name = f"LLE_name={symbol}_window={window}_step={tstep}_rettype={ret_type}_\

d_E={d_E}_tau={tau}_approach={approach_lyap}_max_len={max_len}_separation={sep}.
txt"

np.savetxt(name, LLE)
```

Визначаємо параметри для збереження рисунків:

```
# позначення показника Ляпунова в легенді рисунку
label_lyap = r'$\lambda_{max}$'

# назва рисунку
file_name =
f"LLE_name={symbol}_window={window}_step={tstep}_rettype={ret_type}_\

d_E={d_E}_tau={tau}_approach={approach_lyap}_max_len={max_len}_separation={sep}"

# колір показника
color ='red'
```

та виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          LLE,
          ylabel,
          label_lyap,
          xlabel,
          file_name,
          color)
```

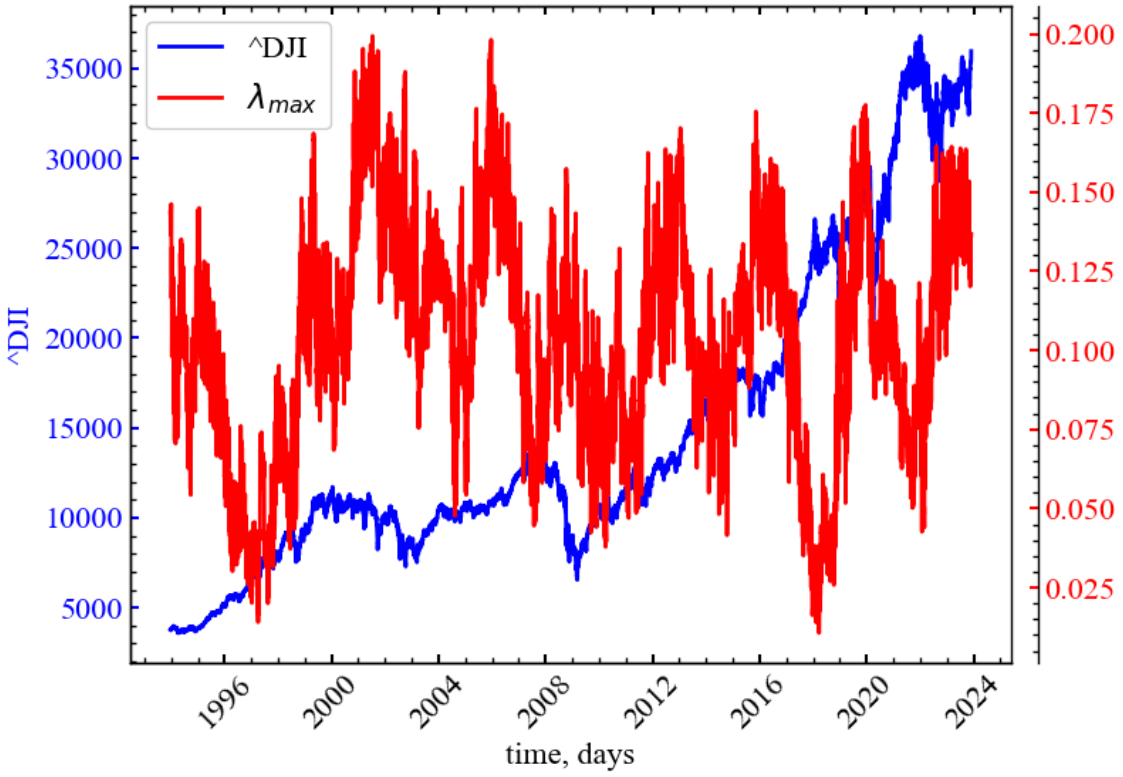


Рис. 9.4: Динаміка індексу Доу Джонса та старшого показника Ляпунова

Бачимо (Рис. 9.4), що СПЛ починає спадати в кризові та передкризові стани, що вказує на зростання корельованності досліджуваної динаміки. У момент кризи СПЛ починає зростати, що вказує на зростання дивергенції в кризові періоди.

9.2.1.2 Обчислення показників Ляпунова на основі методу Екмана

Визначимо наступні параметри:

```
window = 500 # ширина вікна
tstep = 1 # часовий крок вікна
length = len(time_ser)  # довжина самого ряду
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_E = 4 # розмірність вкладень вихідного простору (кількість показників)
d_M = 3 # розмірність вкладень підпростору

approach_lyap = "eckmann1986" # метод для розрахунку старшого показника
sep = "auto" # оцінка середнього періоду як величину, обернену до середньої
частоти спектра потужності
min_neighb = "default" # min(2 * matrix_dim, matrix_dim + 4)

LE = []                          # масив для збереження ПЛ
```

Тепер переходимо до розрахунків:

```
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

    le, _ = nk.complexity_lyapunov(signal=fragm,
                                   method=approach_lyap,
                                   dimension=d_E,
                                   matrix_dim=d_M,
                                   min_neighbors=min_neighb,
                                   separation=sep,
                                   show=False)

    LE.append(le)
```

Зберігаємо отримані результати в текстових файлах:

```
LE = np.array(LE)

for i in range(d_E):
    np.savetxt(f"LE
number={i+1}_name={symbol}_window={window}_step={tstep}_rettype={ret_type}_\

d_E={d_E}_d_M={d_M}_approach={approach_lyap}_min_neighbors={min_neighb}_separati
on={sep}.txt", LE[i])
```

Візуалізуємо отримані результати:

```
fig, ax = plt.subplots(LE.shape[1]+1, 1, sharex=True)

ax[0].plot(time_ser.index[window:length:tstep],
time_ser.values[window:length:tstep], label=symbol)
ax[0].set_ylabel(symbol)
ax[0].legend()

for i in range(1, LE.shape[1]+1):
    ax[i].plot(time_ser.index[window:length:tstep], LE[:, i-1], color='red',
label=fr'$\lambda_{i}$')
    ax[i].set_ylabel(fr"$\lambda_{i}$")
    ax[i].legend()

ax[-1].set_xlabel(xlabel)
fig.subplots_adjust(hspace=0)

plt.savefig(f"LE name={symbol}_window={window}_step={tstep}_rettype={ret_type}_\

#d_E={d_E}_d_M={d_M}_approach={approach_lyap}_min_neighbors={min_neighb}_separat
ion={sep}.jpg")
plt.show();
```

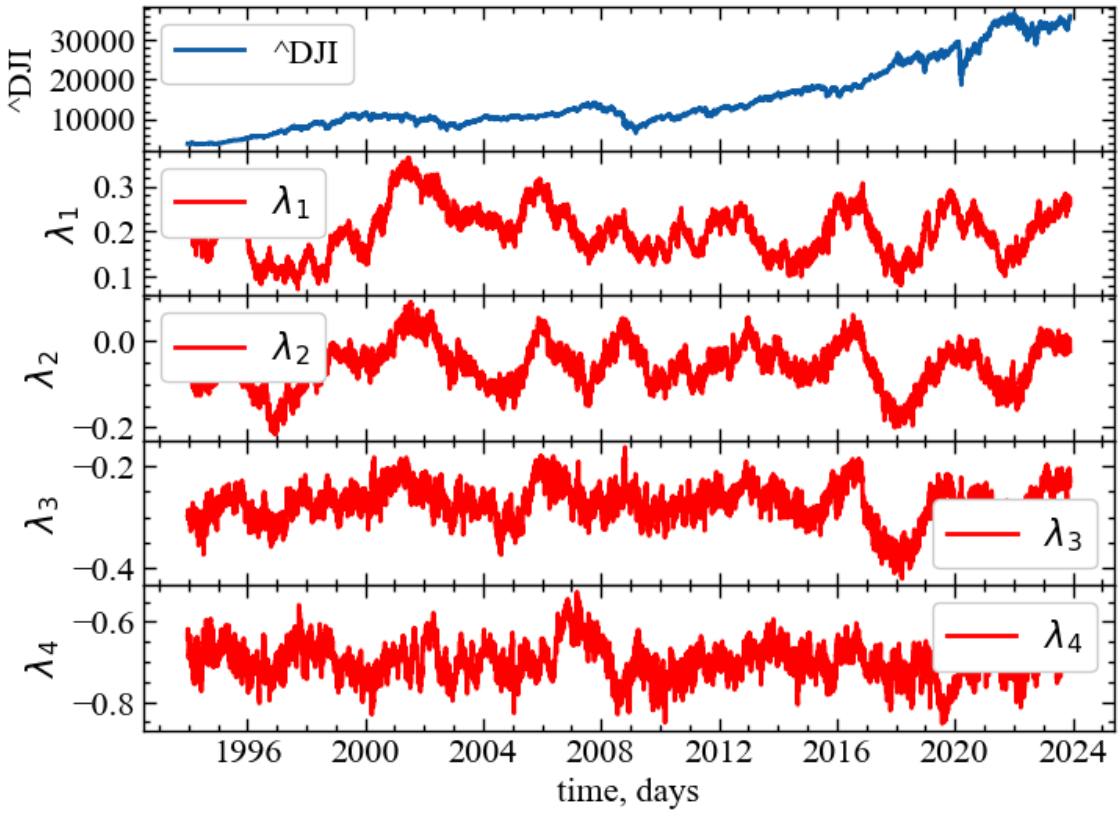


Рис. 9.5: Динаміка індексу Доу Джонса та спектра показників Ляпунова

Як показано на Рис. 9.5, спектр показників Ляпунова реагує особливим чином на кризові події фондового ринку. Видно, що, по-перше, λ спадає в передкризові періоди та зростає під час кризи. Особливо характерною є дана

динаміка перед кризами 1997, 2001, 2008, 2011, 2015, 2020 років. У передкризові періоди спостерігається конвергенція траєкторій у фазовому просторі системі, що говорить про зростання її впорядкованості. Сам кризовий та посткризовий періоди характеризуються дивергенцією, тобто розбіжністю траєкторій системи. По-друге, видно, що, спускаючись від 1-го до 4-го показника Ляпунова, ми поступово втрачаємо інформацію про динаміку системи. Тобто, перші найбільші показники представляються в даному випадку найбільш інформативними. Можливо, у даному випадку, має сенс розглядати лише старший ПЛ.

На Рис. 9.6 представлено порівняльну динаміку індексу Доу-Джонса та старшого показника Ляпунова на основі методу Екмана.

Зберігаємо показник у текстовому файлі:

```
name = f"LE Eckman
name={symbol}_window={window}_step={tstep}_rettype={ret_type}_\

#d_E={d_E}_d_M={d_M}_approach={approach_lyap}_min_neighbors={min_neighb}_separat
ion={sep}.txt"

np.savetxt(name, LE[:, 0])
```

Визначаємо параметри для збереження рисунків:

```
# позначення показника Ляпунова в легенді рисунку
label_lyap = r'$\lambda_{max}$'

# назва рисунку
file_name = f"LE Eckman
name={symbol}_window={window}_step={tstep}_rettype={ret_type}_\

#d_E={d_E}_d_M={d_M}_approach={approach_lyap}_min_neighbors={min_neighb}_separat
ion={sep}"

# колір показника
color ='red'
```

Візуалізуємо старший показник Ляпунова:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          LE[:, 0],
          ylabel,
          label_lyap,
          xlabel,
          file_name,
          color)
```

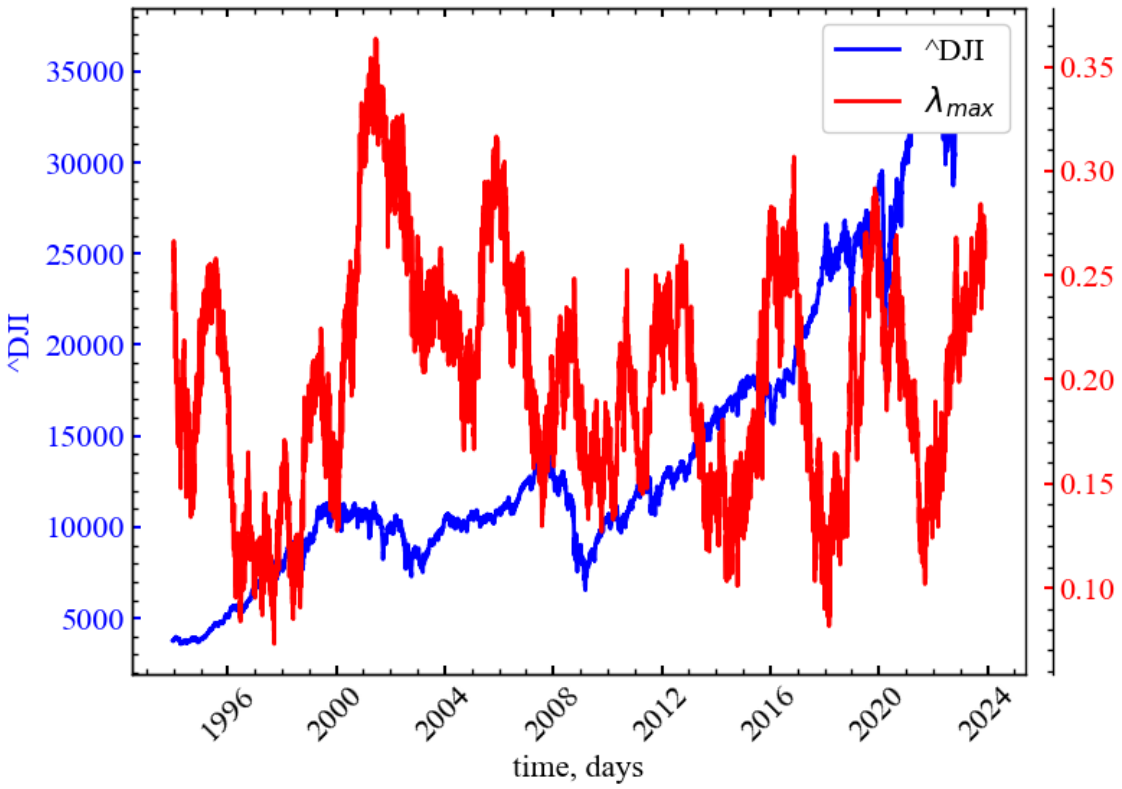


Рис. 9.6: Динаміка індексу Доу Джонса та старшого показника Ляпунова отриманого за допомогою методу Екмана

9.3 Висновок

Теорія хаосу та її інструментарій залишаються величезним викликом для дослідників різних галузей науки. У світі показників Ляпунова зберігається зростаючий інтерес до їх визначення, чисельних методів та застосування до різних складних систем. Підсумовуючи, СПЛ дозволяє встановити:

- область чутливості до початкових умов;
- область хаосу;
- область стабільності.

9.4 Завдання для самостійної роботи

1. Оберіть часовий варіант згідно вашого варіанту
2. Визначіть період прогнозованості досліджуваної системи за діаграмою розбіжності фазових траєкторій
3. Проаналізуйте динаміку старшого показника Ляпунова та його спектру за описаними в лабораторній роботі алгоритмами

9.5 Контрольні запитання

1. Поясніть, які відмінності хаотичного ряду динаміки від інших процесів.
2. Яким чином наявність або відсутність хаотичності може впливати на можливість прогнозування розвитку певного процесу?

10. Лабораторна робота № 10

Тема. Дослідження складних систем із використанням інструментарію неекстенсивної статистики

Мета. Оволодіти методологією та інструментарієм неекстенсивної статистики стосовно критичних і кризових явищ

10.1 Теоретичні відомості

10.1.1 Неекстенсивна термодинаміка і кризи фінансово-економічних систем

Великий виклик теорії складності, що лежить в основі сучасної наукової парадигми, бере початок ще із старих та таких важливих проблем, як: стріла часу, існування простого та фундаментального фізичного рівня для єдиного опису макроскопічного та мікроскопічного рівнів, взаємозв'язок між спостерігачем та досліджуваним об'єктом, і т. д. Загалом, що стосується теорії складності та кожного нового рівня реальності, потрібні нові концепції та нові класифікації.

Зокрема, теорія складності включає: хаотичну динаміку в просторі станів, далеку від рівноважних фазових переходів, довготривалі кореляції, самоорганізацію та мультимасштабність, фрактальні процеси в просторі і часі та інші значущі явища [152]. Теорія складності розглядається як третя наукова революція минулого століття (після теорії відносності та квантової теорії). Однак теорія складності ще далека від своєї академічної зрілості. У цьому напрямку вагомий внесок щодо питання "що таке складність" можна знайти в книзі Г. Ніколіса та І. Пригожина [153]. Як правило, ми можемо узагальнити основну концепцію теорії складності наступним чином:

1. Теорія складності — це узагальнення статистичної фізики для критичних станів термодинамічної рівноваги та для далеких від рівноваги процесів.
2. Складність — це поширення динаміки на нелінійність і дивну динаміку.
3. Також, згідно Іллі Пригожину, теорія складності пов'язана з динамікою кореляцій замість динаміки траєкторій або хвильових функцій.

Згідно з теорією складності, різні фізичні явища, що відбуваються в розподілених фізичних системах, таких як космічна плазма, рідини або тверді тіла, хімія, біологія, екосистеми, динаміка ДНК, соціально-економічні чи

інформаційні системи, мережі можна описати і зрозуміти подібним чином. Цей опис базується на принципі максимізації ентропії. Також згідно з теорією складності, вказані системи є цілісно стійкими дисипативними структурами, що утворюються загальним природним процесом, спрямованим на максимізацію ентропії. З точки зору складності, немає суттєвої диференціації між групою галактик, зірками, тваринами, квітами або елементарними частинками, оскільки скрізь ми маємо відкриті, динамічні та самоорганізовані системи і всюди природа працює з метою максимізації ентропії.

Під час дослідження складних фізичних систем та явищ, зокрема, самоорганізаційних і фрактальних структур, субдифузії, турбулентності, хімічних реакцій, а також різних економічних, соціальних і біологічних систем розподіл Гіббса не забезпечує узгодження із спостережуваними явищами. Як виявляється у багатьох дослідженнях, для таких систем характерні степеневі розподіли [154]. Вони не отримуються з принципу максимуму ентропії Гіббса-Шеннона, на якому ґрунтується як **рівноважна**, так і **нерівноважна статистична термодинаміка** [155–157]. Це спричинило численні спроби побудови узагальненої статистики, яка б забезпечила степеневу асимптотику функції розподілу. Таку узагальнену статистику можна будувати на основі кількох ентропій. Серед них важливе місце посідає **ентропія Тсалліса** (Tsallis entropy).

Дослідження в області механіки **неекстенсивних (неадитивних)** систем стали останнім часом предметом значного інтересу в зв'язку з проявами неаддитивних властивостей в аномальних фізичних явищах. Це пояснюється як новизною виникаючих тут загальнотеоретичних проблем, так і важливістю практичних застосувань (див. бібліографію, представлену за посиланням (https://tsallis.cbpf.br/biblio.htm), яка постійно оновлюється). Початок систематичного вивчення в цьому напрямку пов'язаний з роботою К. Тсалліса, в якій автором була введена параметрична формула статистичної q-ентропії, залежної від деякого дійсного числа q (так званого параметра деформації) і неадитивної для сукупності незалежних складних систем. Теорія неекстенсивних систем, заснована на ентропії Тсалліса, в даний час інтенсивно розвивається. Ці роботи стали значним кроком у розвитку теоретико-інформаційного підходу і при розробці принципів **неекстенсивної статистичної механіки** та рівноважної термодинаміки відкритих систем. При цьому важливо відзначити, що діапазон застосування цих та багатьох інших неекстенсивних параметричних ентропій в даний час постійно розширюється, охоплюючи різні напрямки в науці, такі як космологія і космогонія, теорія плазми, квантова

механіка і статистика, нелінійна динаміка і фрактали, геофізика, біомедицина і багато інших.

Економічну динаміку з фізичної точки зору можна розглядати як просторово розподілену динаміку та пов'язану із загальною категорією нелінійних розподілених систем. Аналіз економічних часових рядів демонструє складну та хаотичну динаміку у фазовому просторі. Теорема Такенса (за допомогою методу затримок) дозволяє реконструювати топологічний еквівалент до вихідного фазового простору, який зберігає основні геометричні та динамічні властивості, такі як ступені свободи, фрактальна розмірність, мультифрактальність, показники Ляпунова, матриця прогнозування тощо. Реконструйований фазовий простір може бути використаний для оцінки всіх вищезазначених величин, а також фазових переходів, статистичної поведінки, генерування ентропії тощо. Крім того, фазовий простір може мати мультифрактальні властивості та характеристики переривчастої турбулентності, які вказують на існування дальніх взаємодій у просторі та часі, а також мультимасштабну взаємодію.

Ці характеристики також вказують на існування дробової динаміки у фазовому просторі, яку можна описати за допомогою дробово-диференціальних рівнянь Фоккера-Планка та аномальних дифузійних рівнянь. Рішеннями цих рівнянь є дробові просторово-часові функції та негаусові функції розподілу, які належать до категорії розподілів Леві та розподілів Тсалліса. Нерівноважні стаціонарні стани економічної динаміки походять від процесів сильної самоорганізації, що відповідає локальним максимумам ентропії Тсалліса, тоді як зміни параметрів управління економічної системи можуть спричинити фазовий перехід та зміщення економічної динаміки до нової стійкої рівноваги, стійкого стану з максимальною ентропією Тсалліса. Цей фазовий перехід призводить до мультифрактальної зміни у формуванні фазового простору та до зміни феноменології економічної системи. Нарешті, статистику динаміки в мультифрактальному фазовому просторі можна описати за допомогою степеневих функцій розподілу Тсалліса з "важкими" хвостами, які можуть бути використані для вдосконалення методів прогнозування.

В останні роки статистична механіка розширила своє початкове призначення: застосування статистики до великих систем, стани яких регулюються якимись гамільтоновими функціоналами [154]. Їх здатність пов'язувати мікроскопічні стани окремих складових системи з макроскопічними властивостями сьогодні використовується повсюдно [155]. Безумовно, найважливішим із цих зв'язків все-таки є визначення термодинамічних

властивостей через відповідність між поняттям ентропії, спочатку введеним Рудольфом Клаузіусом в 1865 р., та кількістю дозволених мікроскопічних станів, введеним Людвігом Больцманом близько 1877 р. коли він вивчав підхід до рівноваги ідеального газу [157]. Цей зв'язок можна виразити як

$$S = k \ln W, \tag{10.1}$$

де k — позитивна константа, а W — кількість мікростанів, сумісних з макроскопічним станом ізольованої системи. Це рівняння, відоме як принцип Больцмана, є одним із наріжних каменів стандартної статистичної механіки. Коли система не ізольована, а замість цього контактує з деяким великим резервуаром, можна модифікувати рівняння (10.1) і отримати ентропію Больцмана-Гіббса (БГ-BG):

$$S_{BG} = -k \sum_{i=1}^{W} p_i \ln p_i, \tag{10.2}$$

де p_i — ймовірність мікроскопічної конфігурації i [154]. Статистична механіка BG все ще ґрунтується на таких гіпотезах, як молекулярний хаос [153] та ергодичність [158]. Незважаючи на відсутність фактичного фундаментального виведення, статистика BG, безсумнівно, мала успіх у вивченні систем, в яких домінують короткі просторово-часові взаємодії. Отже, цілком можливо, що інші фізичні ентропії, крім BG, можуть бути визначені для належного опису аномальних систем, для яких спрощена гіпотеза про ергодичність та/або незалежність не виконується. Натхненний такими концепціями в 1988 р. Константіно Тсалліс (Constantino Tsallis) запропонував узагальнення статистичної механіки BG, яка охоплює системи, що порушують ергодичність, системи, мікроскопічні конфігурації яких не можна вважати незалежними. Це узагальнення базується на неадитивних ентропіях, S_q, що характеризується індексом q і призводить до неекстенсивної статистики

$$S_q = -k \left(1 - \sum_{i=1}^{W} p_i^q\right) \Big/ (1-q), \tag{10.3}$$

p_i — ймовірності, пов'язані з мікроскопічними конфігураціями, W — їх загальне число, q — дійсне число, і k — постійна Больцмана. Значення q є мірою неекстенсивності системи. При цьому, $q = 1$ відповідає стандартній статистиці BG. Вираз (10.3) модифікує S_{BG} ($\lim q \to 1, S_q = S_{BG}$), як основу

можливого узагальнення статистичної механіки BG [159,160]. Значення ентропійного індексу q для конкретної системи повинно визначатися апріорі з мікроскопічної динаміки.

З часу своєї появи ентропія Тсалліса (10.3) стала джерелом кількох важливих результатів як у фундаментальній, так і в прикладній фізиці, а також в інших наукових областях, таких як біологія, хімія, економіка, геофізика та медицина [161].

10.1.2 Неекстенсивна ентропія і триплет Тсалліса

Системи, що характеризуються статистичною механікою Больцмана-Гіббса, мають такі характеристики: (i) їх функції розподілу для енергій пропорційні експоненціальній функції; (ii) вони мають сильну чутливість до початкових умов, яка з часом зростає в геометричній прогресії (хаос), характеризуючись позитивним максимальним показником Ляпунова; (iii) їх релаксація відбувається експоненційно з певним часом релаксації. Іншими словами, ці три способи поведінки описуються експоненціальними функціями (тобто $q = 1$). Однак встановлено, що для систем, які можна вивчати в рамках неекстенсивної статистичної механіки, функція щільності ймовірності енергії (пов'язана зі стаціонарністю або рівновагою), чутливість до початкових умов та релаксація описуються трьома ентропійними індексами $q_{stat}, q_{sens}, q_{rel}$, які отримали назву **триплета Тсалліса** (Tsallis triplet), або q-триплета Тсалліса [155,162].

Неекстенсивна статистична теорія математично базується на нелінійному рівнянні

$$\frac{dy}{dx} = y^q, \tag{10.4}$$

розв'язком якого є q-експоненціальна функція

$$\exp_q(x) = \begin{cases} (1 + (1-q)x)^{1/(1-q)}, & \text{якщо } 1 + (1-q)x > 0, \\ 0, & \text{якщо } 1 + (1-q)x \leq 0. \end{cases} \tag{10.5}$$

Для $q \to 1q$-Гаусіан відповідає звичайному розподілу Гауса.

Розв'язок рівняння (10.4) можна реалізувати трьома різними способами, включеними до q-триплету Тсалліса: $(q_{sens}, q_{stat}, q_{rel})$. Ці величини характеризують три фізичні процеси, які узагальнені тут, тоді як значення q-триплету характеризують атракторний набір динаміки у фазовому просторі

динаміки, і вони можуть змінюватися, коли динаміка системи притягується до іншого набору атракторів.

Для неекстенсивної системи величина q-індексу залежить від оцінюваних властивостей динаміки і фазового простору системи. Для динамічних систем оцінюється q–триплет, що відображає три властивості системи (Рис. 10.1). Індекс q_{stat} оцінюється на основі рівноважної моделі рангового розподілу з використанням методів нелінійного оцінювання [163]. Цей індекс є параметром області атракції системи. Індекс q_{sens} відображає чутливість системи до початкових умов та виробництво ентропії і визначається за мультифрактальним спектром [164]. Релаксаційний індекс q_{rel} знаходять на основі автокореляції і характеристик дифузійних процесів [165].

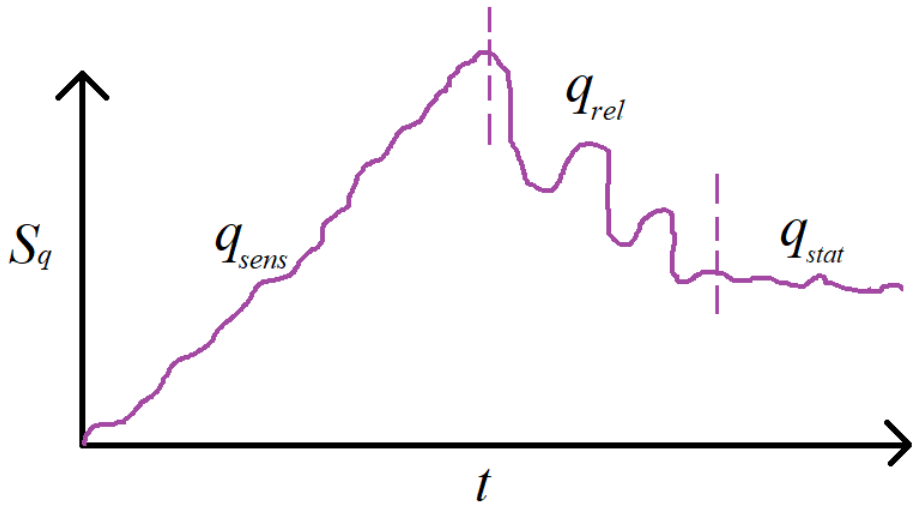


Рис. 10.1: Часова періодизація періодів виробництва q-ентропії. Перший період відповідає виробництву ентропії через параметр q_{sens} q-триплету Тсалліса. Другий період відповідає певному процесу релаксації через параметр q_{rel}. Система виявляє коливання через параметр q_{stat} q-триплету Тсалліса

10.1.2.1 Стаціонарність $q = q_{stat}$

Значення q для стаціонарного стану оцінюють із функції розподілу прибутковостей, що в свою чергу отримується шляхом підгонки q-Гаусіана:

$$P_q(\beta, x) = \left(\sqrt{\beta}/C_q\right) \exp(-\beta r x^2) \tag{10.6}$$

для емпірично побудованої гістограми $\{p(x_i) \mid i = 1, \ldots, N\}$ та різних значень β, що підбираються шляхом мінімізації $\sum_i \left[P_{q_{stat}}(\beta, x_i) - p(x_i)\right]^2$. Залежно від значення q, C_q може приймати наступні види:

$$C_q =$$

$$\begin{cases} 2\sqrt{\pi}\,\Gamma\left(\dfrac{1}{1-q}\right)/(3-q)\sqrt{1-q}\,\Gamma\left(\dfrac{3-q}{2(1-q)}\right), & \text{якщо } -\infty < q < 1, \\ \sqrt{\pi}, & \text{якщо } q = 1, \\ \sqrt{\pi}\,\Gamma\left(\dfrac{3-q}{2(q-1)}\right)/\sqrt{q-1}\,\Gamma\left(\dfrac{1}{q-1}\right), & \text{якщо } 1 < q < 3. \end{cases} \tag{10.7}$$

Для оцінки динаміки значення q будується графік залежності $\ln_q[p(x)]$ від x^2 для вибраного інтервалу q (наприклад, від 1 до 5), що забезпечує найкраще лінійне наближення (оцінюється за максимальним коефіцієнтом детермінації R^2) [166]. Зрозуміло, що значення $p(x)$ стають помітно негаусівськими вздовж хвостів, і замість цього можуть бути описані степеневим законом.

10.1.2.2 Релаксація $q = q_{rel}$

Відповідне q-значення для релаксаційного процесу знаходиться з коефіцієнта автокореляції:

$$C(\tau) = \sum_t |g_{t+\tau}| \cdot |g_t| \Big/ \sum_t |g_t|^2. \tag{10.8}$$

Для статистики BG така кореляція має спадати експоненціально. Той самий алгоритм, що й для q_{stat}, необхідно проробити на графіку залежності $\ln_q[C(\tau)]$ від τ щоб визначити, який набір q найкраще лінеаризує емпіричні дані.

10.1.2.3 Чутливість до початкових умов $q = q_{sens}$

Виробництво ентропії пов'язане із загальним характером атракторної множини. Цей атрактор може бути описаний мультифрактальністю, а також чутливістю до початкових умов. Чутливість до початкових умов можна виразити як

$$\frac{d\xi}{dt} = \lambda_1 \xi + (\lambda_q - \lambda_1)\xi^q, \tag{10.9}$$

де ξ — відхилення траєкторії у фазовому просторі: $\xi \equiv \lim_{\delta \to 0}[\delta(t)/\delta(0)]$, і $\delta(t)$ — це відстань між сусідніми траєкторіями через час t. Розв'язок рівняння (10.9) може бути представлений у вигляді:

$$\xi = \left[1 - \left(\frac{\lambda_{q_{sens}}}{\lambda_1}\right) + \left(\frac{\lambda_{q_{sens}}}{\lambda_1}\right) exp\big((1 - q_{sens})\lambda_1 t\big)\right]^{1/(1-q_{sens})}. \quad (10.10)$$

Спочатку було висловлено гіпотезу, а згодом доведено для часових рядів неекстенсивних систем різної природи, що має місце таке співвідношення [167]:

$$1/(1 - q_{sens}) = 1/\alpha_{min} - 1/\alpha_{max}, \quad (10.11)$$

де α_{min} та α_{max} — відповідно мінімальні та максимальні значення α відповідного мультифрактального спектру $f(\alpha)$.

Спектр мультифрактальності, в свою чергу, випливає з процедури мультифрактального аналізу детрендованих флуктуацій (МФ-АДФ), що дозволяє розрахувати показник Херста для різних часових масштабів.

10.2 Хід роботи

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

Спочатку імпортуємо необхідні бібліотеки:

```
import matplotlib.pyplot as plt
import numpy as np
import yfinance as yf
import pandas as pd
import scienceplots
import neurokit2 as nk
import fathon
import scipy
import statsmodels.api as sm
from fathon import fathonUtils as fu
from scipy.stats import norm
from scipy.special import gamma
from scipy.optimize import curve_fit
from tqdm import tqdm

%matplotlib inline
```

Та визначимо необхідні функції для подальшої роботи:

```
# q-експоненціальна функція
def np_exp_q(x, q=1):
    if q == 1:
       return np.exp(x)
    else:
       return (1+(1-q)*x)**(1/(1-q))

# q-логарифм
def np_log_q(x, q=1):
```

```
    if q == 1:
       return np.log(x)
    else:
       return x**(1-q)-1/(1-q)

# значення для обчислення q-гаусіана
def C_q(q=1.0):
    if q == 1:
       return np.sqrt(np.pi)
    elif q < 1:
       return 2*np.sqrt(np.pi)*gamma(1/(1-q))/(3-q)*np.sqrt(1-q)*gamma((3-
q)/(2*(1-q)))
    elif q > 1:
       return (np.sqrt(np.pi)*gamma((3-q))/(2*(q-1)))/(np.sqrt(q-1)*gamma(1/(q-
1)))

# функція щільності q-гаусіана для обчислення q_stat
def G_q(r, beta, q):
    return np.sqrt(beta)/C_q(q) * np_exp_q(-beta*r, q)

# функція автокореляцій для обчислення q_rel
def acf(x, maxlag):

    n = len(x)
    a = (x - x.mean()) / (x.std() * n)
    b = (x - x.mean()) / x.std()

    cor = np.correlate(a, b, mode="full")
    acf = cor[n:n+maxlag+1]
    lags = np.arange(maxlag +1)

    return acf, lags

# рункція релаксацій для обчислення q_rel
def rel_func(x, q, tau):
    return np_exp_q(-x/tau, q)

# функція для обчислення прибутковостей ряду чи його стандартизації
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

# функція 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', rotation=45, **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();
```

Далі виконаємо налаштування формату виведення рисунків:

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
```

```
'figure.figsize': (8, 6),              # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                     # розмір фонтів рисунку
'lines.linewidth': 2,                  # товщина ліній
'axes.titlesize': 'small',             # розмір титулки над рисунком
'axes.labelsize': size,                # розмір підписів по осям
'legend.fontsize': size,               # розмір легенди
'xtick.labelsize': size,               # розмір розмітки по осі Ox
'ytick.labelsize': size,               # розмір розмітки по осі Ox
"font.family": "Serif",                # сімейство стилів підписів
"font.serif": ["Times New Roman"],     # стиль підпису
'savefig.dpi': 300,                    # якість збережених зображень
'axes.grid': False                     # побудова сітки на самому рисунку
}

plt.rcParams.update(params)            # оновлення стилю згідно налаштувань
```

У цій роботі розглянемо динаміку неекстенсивних показників на прикладі фондового індексу S&P 500, але дивитимемось на ряд, починаючи з 2016 року. Для отримання значень індексу скористаємось бібліотекою `yfinance`.

```
symbol = '^DJI'                        # Символ індексу
start = "2016-01-01"                   # Дата початку зчитування даних
end = "2023-12-01"                     # 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 Ox
ylabel = symbol                        # підпис по вісі Oy
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того, з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'              # підпис по вісі Ox
ylabel = symbol                        # підпис по вісі Oy
```

Виводимо досліджуваний ряд:

```
fig, ax = plt.subplots(1, 1)                    # 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 Ox
ax.set_ylabel(ylabel)                           # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                         # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                    # Зберігаємо графік
plt.show();                                     # Виводимо графік
```

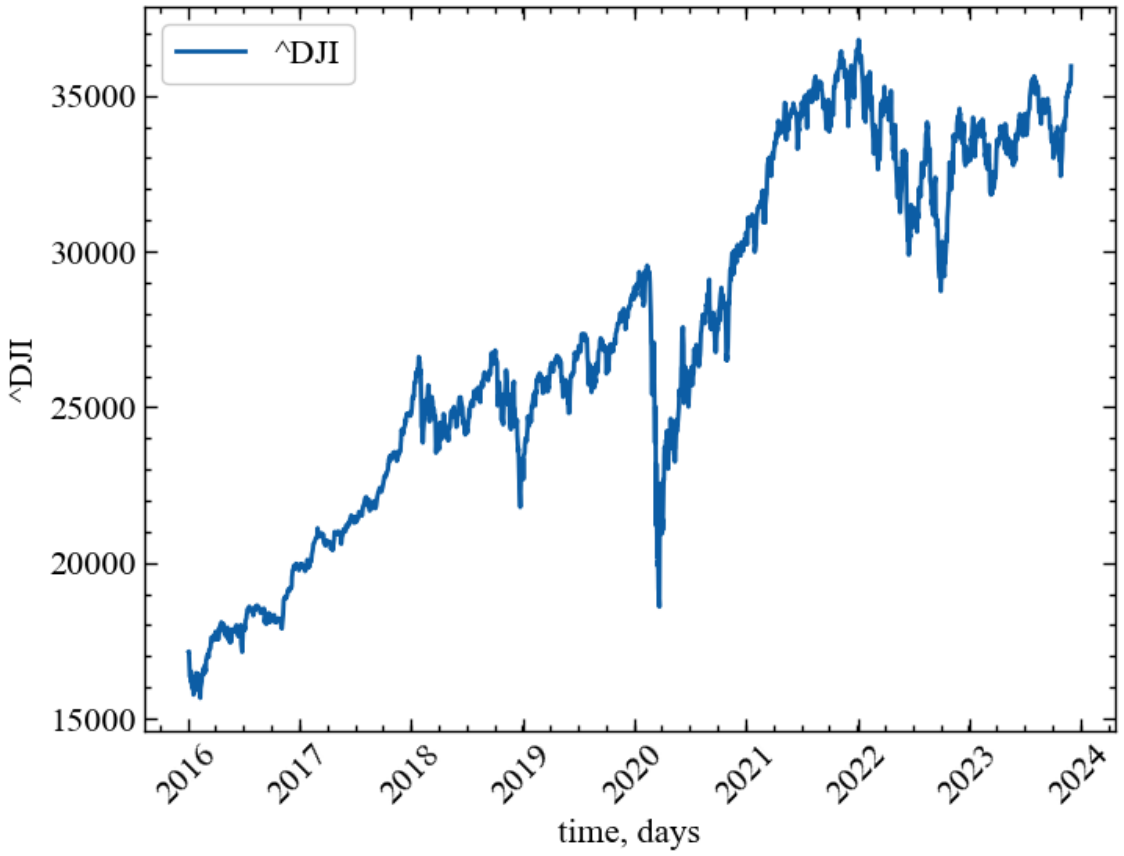


Рис. 10.2: Динаміка щоденних змін фондового індексу Доу Джонса

10.2.1 Розрахунок показника q_{stat}

10.2.1.1 Побудова q-Гаусіана для всього ряду

```
q_stat_time_ser = time_ser.copy()
ret_type = 4 # визначення типу ряду для його перетворення
q_stat_time_ser = transformation(q_stat_time_ser, ret_type)

hist, bin_edg = np.histogram(q_stat_time_ser, bins=250, density=True)

mu, std = norm.fit(q_stat_time_ser)
x = np.linspace(q_stat_time_ser.min(), q_stat_time_ser.max(), len(bin_edg[1:]))
p = norm.pdf(x, mu, std)
```

```
xval = bin_edg[1:]**2
yval = hist

popt, pcov = curve_fit(G_q, xdata=xval, ydata=yval, bounds=([0.0, 0.0], [np.inf,
3.0]))

fig, ax = plt.subplots(1, 1)
ax.plot(bin_edg[1:], hist, 'o', label=r"$P_{емпіричний}$")
ax.plot(x, p, 'o', label="Гаус")
ax.plot(x, G_q(x**2, popt[0], popt[1]), label=r"$q$-Гаусіан")
ax.set_yscale('log')
ax.set_xlabel("x")
ax.set_ylabel(r"$\log{P(\beta, x)}$")

plt.legend()
plt.show();
```

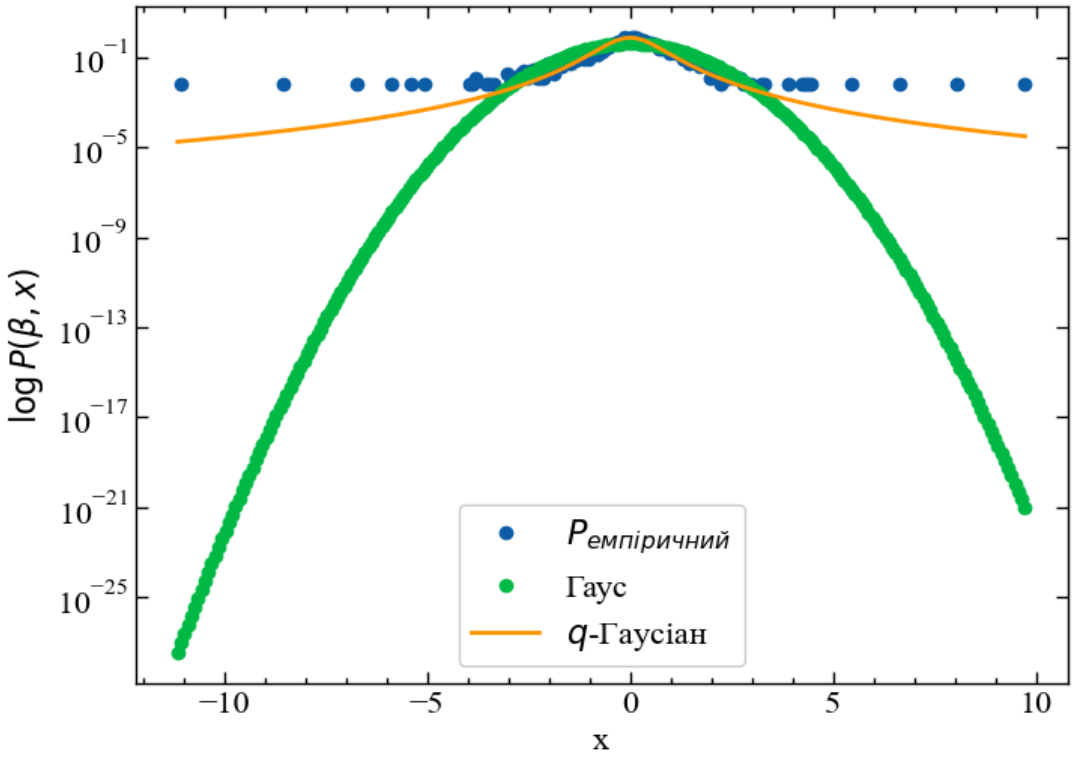


Рис. 10.3: Функція розподілу нормалізованих прибутковостей для Доу Джонса в порівнянні з Гаусіан та q-Гаусіаном

На Рис. 10.3 видно, що стандартизовані прибутковості для індексу Доу Джонса виходять за межі $\pm 10\sigma$. Як можна бачити, теоретичний розподіл Гауса значно недооцінює появу екстремально високих і низьких прибутковостей. Якщо логарифм емпіричної імовірності для таких прибутковостей знаходиться на рівні 10^{-1}, то розподіл Гауса становить приблизно 10^{-25}. Тобто, розподіл Гауса недооцінює еміричну ймовірність у 10^{24} рази. Хоча q-Гаусіан також не представляється ідеальним для опису таких прибутковостей, але недооцінка

важких хвостів у випадку неекстенсивної статистики набагато менша в порівнянні зі звичайним Гаусіаном.

10.2.1.2 Розрахунок q_{stat} у віконній процедурі

```
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

length = len(time_ser)

q_stats = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()  # відбираємо фрагмент

    fragm = transformation(fragm, ret_type)   # виконуємо процедуру
# трансформації ряду

    hist_fragm, bin_edg_fragm = np.histogram(fragm, bins=100, density=True)

    xval = bin_edg_fragm[1:]**2
    yval = hist_fragm

    popt, pcov = curve_fit(G_q, xdata=xval, ydata=yval, bounds=([0.01, 1.0],
[np.inf, 5.0]))
    q_stat = popt[1]


    q_stats.append(q_stat)
```

Зберігаємо отримані результати в текстовому файлі:

```
name =
f"q_stat_name={symbol}_window={window}_step={tstep}_rettype={ret_type}.txt"

np.savetxt(name, q_stats)
```

Визначаємо параметри для збереження рисунків:

```
# позначення показника q_stat в легенді рисунку
label_q_stat = r'$q_{stat}$'

# назва рисунку
file_name =
f"q_stat_name={symbol}_window={window}_step={tstep}_rettype={ret_type}"
```

```
# колір показника
color = 'brown'

plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          q_stats,
          ylabel,
          label_q_stat,
          xlabel,
          file_name,
          color)
```

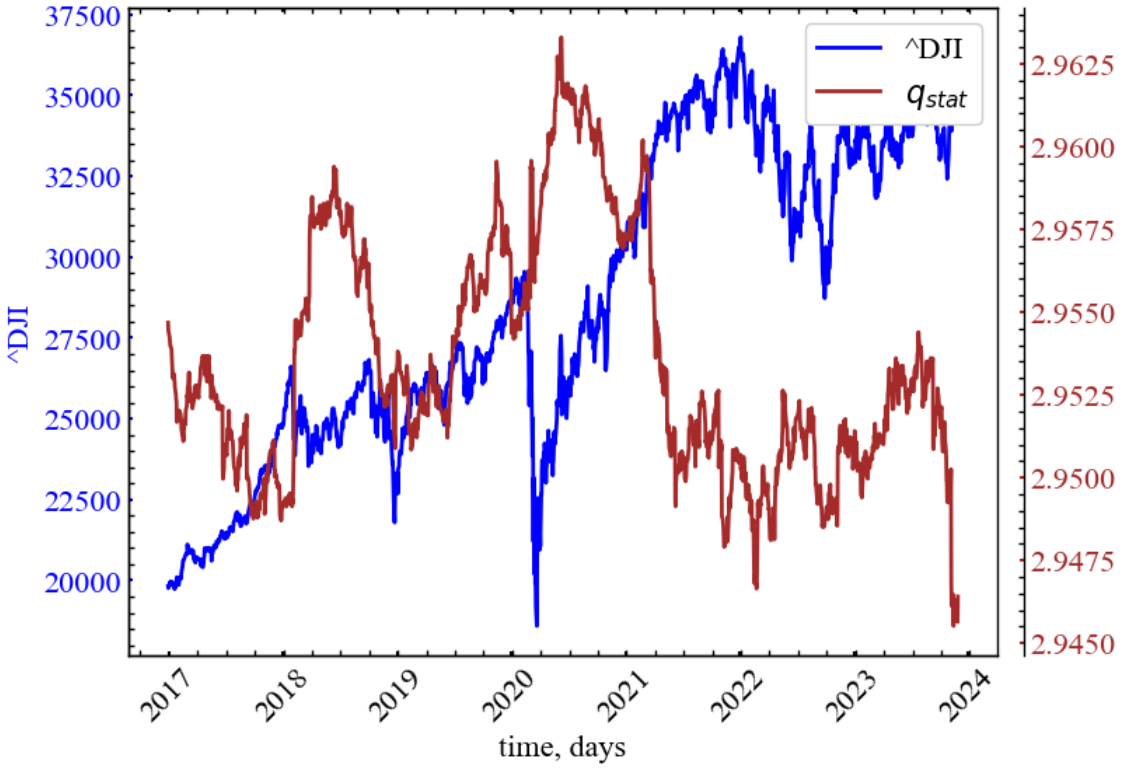


Рис. 10.4: Порівняльна динаміка коливань ціни індексу Доу Джонса та показника q_{stat}

Як ми можемо бачити на Рис. 10.4, показник q_{stat} зростає під час крахових явищ на фондовому ринку. Це вказує на значне зростання ступеня впливу важких хвостів у розподілі прибутковостей досліджуваного індексу.

10.2.2 Розрахунок показника q_{rel}

```
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
```

```
max_lag = 100

length = len(time_ser)

q_rels = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()  # відбираємо фрагмент

    fragm = transformation(fragm, ret_type)   # виконуємо процедуру
# трансформації ряду

    autocor, lags = acf(x=fragm, maxlag=max_lag)
    lags = lags
    autocor = autocor

    popt, pcov = curve_fit(rel_func, xdata=lags[1:], ydata=autocor[1:],
bounds=(1, [np.inf, 10]))
    q_rel = popt[0]

    q_rels.append(q_rel)
```

Зберігаємо отримані результати в текстовому файлі:

```
name =
f"q_rel_name={symbol}_window={window}_step={tstep}_rettype={ret_type}_maxlag={ma
x_lag}.txt"

np.savetxt(name, q_rels)
```

Визначаємо параметри для збереження рисунків:

```
# позначення показника q_rel в легенді рисунку
label_q_rel = r'$q_{rel}$'

# назва рисунку
file_name =
f"q_rel_name={symbol}_window={window}_step={tstep}_rettype={ret_type}_maxlag={ma
x_lag}"

# колір показника
color = 'red'
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          q_rels,
          ylabel,
          label_q_rel,
          xlabel,
          file_name,
          color)
```

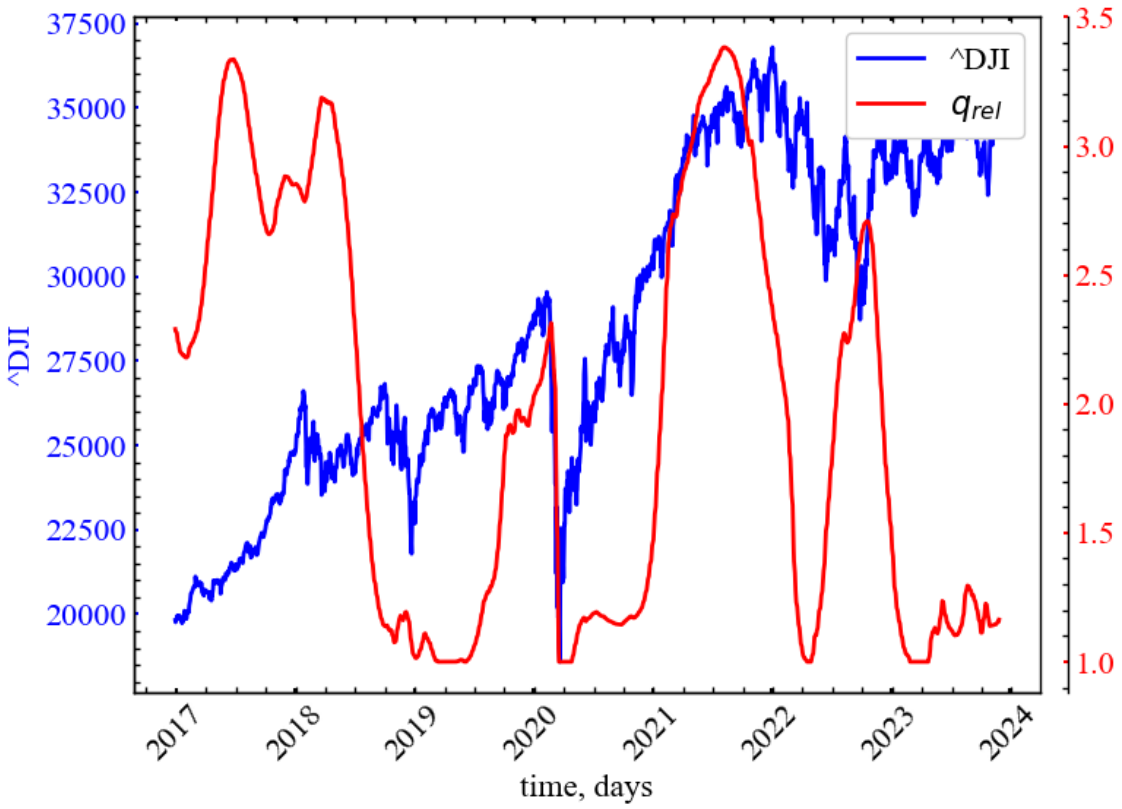


Рис. 10.5: Порівняльна динаміка коливань ціни індексу Доу Джонса та показника q_{rel}

Для досліджуваного показника на Рис. 10.5 видно, що ступінь релаксації зростає саме в передкризовий стан системи, що є індикатором зростання самоорганізації трейдерів через певні зовнішні показники. Дана динаміка узгоджується зі зростанням ступеня автокореляції під час кризових подій, що ми мали змогу спостерігати в першій лабораторній.

10.2.3 Розрахунок показника q_{sens}

```
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

rev = True # Чи повторювати розрахунок ф-ції флуктуацій з кінця
accumulate = False # Повторна акумуляція детрендованого ряду для роботи із
сильно антиколрельованими рядами

q_min = -5 # мінімальне значення q
q_max = 5  # максимальне значення q
```

```
q_inc = 1 # крок збільшення q

win_beg = 10 # Початкова ширина сегменту
win_end = window-1 # Кінцева ширина сегменту

length = len(time_ser)

q = np.arange(q_min, q_max+q_inc, q_inc)
q = np.round_(q, decimals =1)

order = 1 # порядок поліному для детрендування (MF-DFA)

q_sens_values = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()  # відбираємо фрагмент

    fragm = transformation(fragm, ret_type)   # виконуємо процедуру
# трансформації ряду

    if accumulate == True:
        fragm = np.cumsum(fragm-np.mean(fragm))

    a = fu.toAggregated(fragm)

    pymfdfa = fathon.MFDFA(a)

    wins = fu.linRangeByStep(win_beg, win_end)

    n, F = pymfdfa.computeFlucVec(wins, q, revSeg=rev, polOrd=order)
    list_H, list_H_intercept = pymfdfa.fitFlucVec()

    if accumulate == True:
        list_H = list_H - 1

# розрахунок значень tau(q)
    tau = q * list_H - 1

# розрахунок значень сингулярності
    alpha = np.gradient(tau, q, edge_order=2)

# максимальне значення сингулярності
    maximal_alpha = alpha.max()

# мінімальне значення сингулярності
    minimal_alpha = alpha.min()

# розрахунок q_sens
    q_sens = (maximal_alpha-minimal_alpha-
maximal_alpha*minimal_alpha)/(maximal_alpha-minimal_alpha)

    q_sens_values.append(q_sens)
```

Зберігаємо отримані результати в текстовому файлі:

```
name =
f"q_sens_name={symbol}_ret={ret_type}_qmin={q_min}_qmax={q_max}_qinc={q_inc}_win
d={window}_step={tstep}.txt"

np.savetxt(name, q_sens_values)
```

Визначаємо параметри для збереження рисунків:

```
# позначення показника q_rel в легенді рисунку
label_q_sens = r'$q_{sens}$'

# назва рисунку
file_name =
f"q_sens_name={symbol}_ret={ret_type}_qmin={q_min}_qmax={q_max}_qinc={q_inc}_win
d={window}_step={tstep}"

# колір показника
color = 'green'
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          q_sens_values,
          ylabel,
          label_q_sens,
          xlabel,
          file_name,
          color)
```

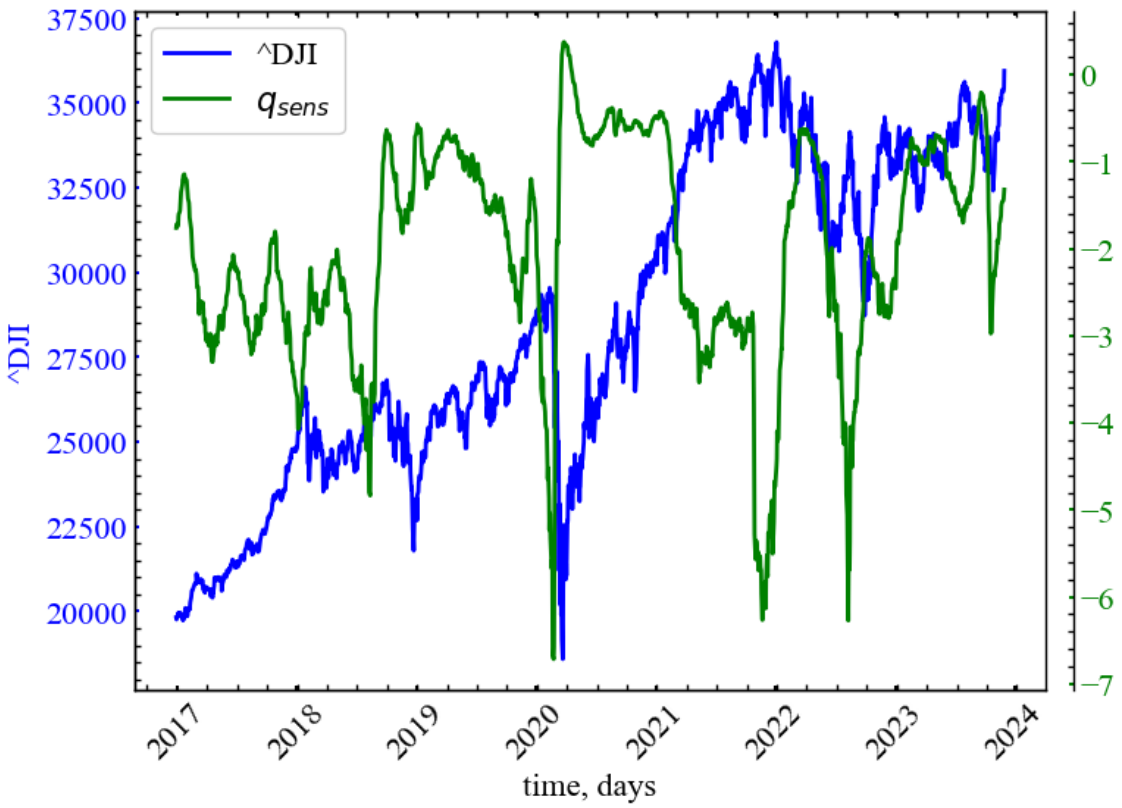


Рис. 10.6: Порівняльна динаміка цін індексу Доу Джонса та показника q_{sens}

Для показника q_{sens} спостерігається спад у передкризові періоди, що вказує на особливу чутливість ринку саме в ці моменти часу. Для повністю ідентичних та незалежно розподілених значень q_{sens} залишався б на рівні 1. У передкризові стани q_{sens} прямує до від'ємних значень, що говорить про конвергенцію атрактора системи до сингулярності, тобто збіжність траєкторій один до одного.

10.2.4 Розрахунок ентропії Тсалліса

```
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)

tsallis_en = []

for i in tqdm(range(0, length-window, tstep)):

    fragm = time_ser.iloc[i:i+window].copy()  # відбираємо фрагмент

    fragm = transformation(fragm, ret_type)   # виконуємо процедуру
# трансформації ряду

    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

    tsen, _ = nk.entropy_tsallis(freq=P,
                                 q=1,
                                 base=np.exp(1))
    tsen /= np.log(len(P))

    tsallis_en.append(tsen)
```

Зберігаємо отримані результати в текстовому файлі:

```
name = f"tsen_name={symbol}_ret={ret_type}_wind={window}_step={tstep}.txt"

np.savetxt(name, tsallis_en)
```

Визначаємо параметри для збереження рисунків:

```
# позначення ентропії Тсалліса в легенді рисунку
label_ts_en = r'$TsEn$'

# назва рисунку
file_name = f"tsen_name={symbol}_ret={ret_type}_wind={window}_step={tstep}"

# колір показника
color = 'purple'
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          tsallis_en,
          ylabel,
          label_ts_en,
          xlabel,
          file_name,
          color)
```

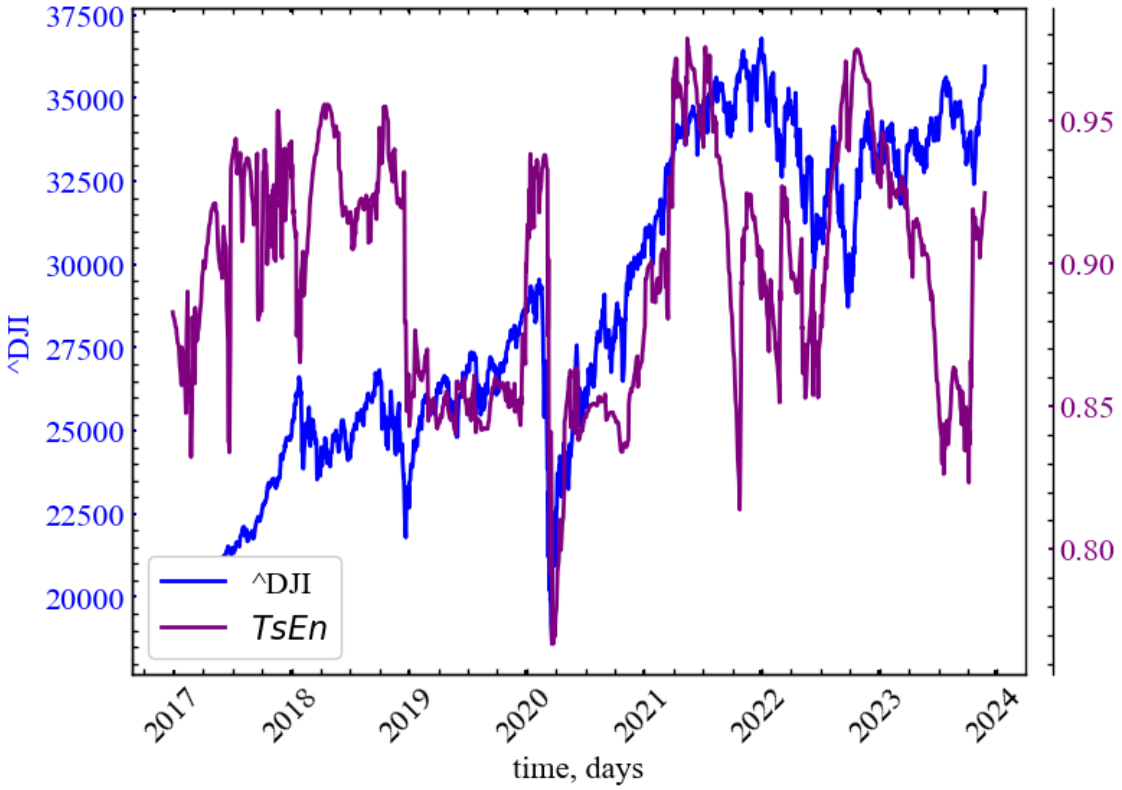


Рис. 10.7: Порівняльна динаміка коливань ціни індексу Доу Джонса та ентропії Тсалліса

З Рис. 10.7 видно, що неекстенсивна ентропія Тсалліса спадає в передкризові періоди, що вказує на зростання ступеня неадитивності (самоорганізованої динаміки) ринку.

10.3 Висновок

У даній лабораторній роботі було представлено неекстенсивний підхід статистичної механіки до динаміки щоденних історичних значень ціни Доу Джонса та його прибутковостей. Встановлено, що індекс Доу Джонса підпорядковується статистиці Тсалліса. Було промодельовано часову динаміку q-триплету, що дало можливість при співставленні з вихідним часовим рядом отримати реакцію компонентів триплету на формування та протікання кризових явищ. Величина q_{stat} у періоди криз зростає, оскільки зростають власне цінові флуктуації. Значення q_{rel} зростає у передкризові періоди, що, очевидно, зумовлено переходом системи в нерівноважний стан і подальшою релаксацією. Нарешті, q_{sens} має мінімальне значення в передкризовий період, вказуючи на особливу чутливість системи поблизу точки біфуркації, якою і є сама криза.

Перспективним представляється дослідження особливостей q-триплету для складних мережних структур, що отримуються при перетворенні часового ряду в мережу одним з відомих методів. Цікавим також є пошук альтернативних компонентів неекстенсивності як, наприклад, міри незворотності часового ряду, чи міри рекурентності тощо. Очевидно, що означені підходи можуть забезпечити необхідний прогрес як на фундаментальному, так і на прикладному рівнях щодо досягнення більш глибокого розуміння природи складних систем.

10.4 Завдання для самостійного виконання

1. Оберіть часовий ряд згідно вашого варіанту
2. Побудуйте емпіричний розподіл прибутковостей вашого ряду та теоретичні адитивний і неекстенсивний розподіли Гауса
3. Побудуйте та проаналізуйте динаміку триплету Тсалліса та неекстенсивної ентропії для кризових подій
4. Сформуйте звіт і зробіть висновки

11. 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

11.1 Теоретичні відомості

Складні системи — це відкриті системи, які обмінюються енергією, речовиною та інформацією з навколишнім середовищем. Досліджуючи складні системи в природничих науках, Пригожин [168] зробив фундаментальне узагальнення, вказавши на необхідність розгляду явищ незворотності та нерівноважності, які є фундаментальними властивостями складних систем різної природи: соціальні, економічні, біомедичні тощо [169]. Пригожин вважав, що найважливіші зміни в сучасній науковій революції пов'язані зі зняттям попередніх обмежень у науковому розумінні часу. Нелінійний світ характеризується рисами темпоральності, тобто незворотністю і швидкоплинністю процесів і явищ. Самоорганізація розглядається як спонтанний процес формування складних систем, що інтегруються. Саме через неоднозначність вибору в точках біфуркації час у теоріях самоорганізації стає справді незворотнім. На відміну від лінійних динамічних теорій — класичної, релятивістської, квантової (де час обернений), в термодинаміці дисипативних структур, створеної Пригожиним, час перестає бути простим параметром і стає поняттям, що виражає темп і напрямок розвитку подій.

Незворотність часу є фундаментальною властивістю нерівноважних дисипативних систем, і її втрата може свідчити про розвиток деструктивних процесів [45,169].

З огляду на статистичні властивості досліджуваного сигналу, його еволюцію можна було б назвати **незворотною** (irreversible), якби була відсутня інваріантність, тобто був би отриманий той же сигнал, якби ми виміряли його в протилежному напрямку. Функція f може бути застосована для знаходження характеристик, які відрізняються прямою і зворотною версіями, тобто часові ряди незворотні, якщо $f(X^d) \neq f(X^r)$. Основна ідея цього визначення полягає в тому, що немає ніяких обмежень на $f(\cdot)$.

Передбачається, що стаціонарний процес X називається **статистично зворотним у часі**, якщо розподіл імовірностей прямої та зворотної систем

приблизно однаковий [170–172]. Незворотність часових рядів вказує на наявність нелінійних залежностей (пам'яті) [173] у динаміці далекої від рівноваги системи, включаючи негаусові випадкові процеси та дисипативний хаос.

У першій групі методів виконується символізація часових рядів, а потім проводиться аналіз шляхом статистичного порівняння рядка символів у прямому та зворотному напрямках [174]. Іноді використовуються додаткові алгоритми стиснення [175]. Важливим кроком для цієї групи є символізація — перетворення часового ряду у символьний ряд вимагає додаткової спеціальної інформації (наприклад, поділ діапазону або розмір алфавіту) і, отже, містить проблему залежності алгоритму від цих додаткових параметрів. Друга проблема виникає при розгляді масштабної інваріантності складних сигналів. Оскільки процедури типових символізацій є локальними, врахування різних масштабів може викликати певні труднощі [45].

Інша група методів формалізації показника незворотності не використовує процедуру символізації, а базується на використанні вихідних значень часових рядів або прибутковостей.

Один з таких підходів базується на асиметрії розподілу точок на **діаграмі Пуанкаре** (Poincaré diagram), побудованої на основі значень досліджуваного часового ряду [176,177].

Нещодавно було запропоновано принципово новий підхід до вимірювання незворотності часових рядів, який використовує методи теорії складних мереж [170,178] та поєднує два інструменти: алгоритм графа видимості, що перетворює часові ряди у складну мережу та алгоритм дивергенції Куллбака-Лейблера [178]. Перший формує спрямовану мережу за геометричним критерієм. Потім ступінь незворотності ряду оцінюється за розбіжністю Кульбака-Лейблера. Цей метод є обчислювально ефективним, не вимагає спеціального процесу символізації і, на думку авторів, природно враховує мультимасштабність.

11.1.1 Незворотність на основі діаграм Пуанкаре

Діаграма Пуанкаре для часового ряду являє собою графік, на осі x якого розташовані значення для поточного часу t, а на осі y — його наступні значення в часі $t + \tau$. Усі наступні значення, які рівні один одному $\big(x(t) = x(t + \tau)\big)$, розташовані на **лінії ідентичності** (line of identity, LI). Інтервали, що представляють зростаючу тендецію, відмічені вище LI $\big(x(t) < x(t + \tau)\big)$, тоді як

спадна тенденція характеризуватиметься скупченням точок нижче LI $\big(x(t) > x(t+\tau)\big)$. Оцінюючи асиметрію точок на діаграмі, ми можемо вивести різні кількісні показники незворотності (асиметрії) досліджуваних систем [179,180].

11.1.1.1 Індекс Гузіка

Індекс Гузіка (Guzik index, GIx) [176] можна визначити як відношення відстаней точок вище LI до відстаней усіх точок на діаграмі:

$$GIx = \sum_{i=1}^{a} (D_i^+)^2 \Big/ \sum_{i=1}^{m} (D_i)^2,$$

де $a = C(P_i^+)$ позначає кількість точок над LI; $m = C(P_i^+) + C(P_i^-)$ позначає кількість точок на діаграмі Пуанкаре; D_i^+ це відстань від точки над LI до самої LI. Відстань точки до LI можна визначити як

$$D_i = (|x(i+\tau) - x(i)|)/\sqrt{2}.$$

11.1.1.2 Індекс Порти

Індекс Порти (Porta index, PIx) [177] визначається як кількість точок нижче LI, поділена на загальну кількість точок на діаграмі Пуанкаре, за винятком тих, що знаходяться на LI:

$$PIx = b/m,$$

де $b = C(P_i^-)$ кількість точок нижче LI.

11.1.1.3 Індекс Кошти

Індекс Кошти (Costa index, CIx) [181] враховує кількість приростів $(x(i+1) - x(i) > 0)$ та спадів $(x(i+1) - x(i) < 0)$. Вони представляються симетричними, якщо рівні один одному. Даний індекс розраховується для двовимірної мультимасштабної площини $\big(x(i), x(i+L)\big)$, де новий крос-гранульований ряд $y_\tau(i) = x(i+L) - x(i)$ для $1 \le i \le N - \tau$ відображає асиметрію приростів та спадів ряду, й індекс незворотності для діапазону масштабів τ визначається виразом:

$$CIx_\tau = \left(\sum\nolimits_{y_\tau<0} H[y_\tau] - \sum\nolimits_{y_\tau>0} H[y_\tau]\right)/(N-\tau).$$

Узагальнений CIx для діапазону мастабів τ може бути визначений як

$$CIx = \frac{1}{L}\sum_{\tau=1}^{L}|CIx_\tau|,$$

де L — це максимальний масштаб.

11.1.1.4 Індекс Ейлера

Опираючись на асиметрію розподілу точок нижче та вище LI, Ейлер (Ehler) запропонував наступний індекс асиметрії [182]:

$$EIx = \sum_{i=1}^{N-1}[x(i) - x(i+\tau)]^3 \Bigg/ \left(\sum_{i=1}^{N-1}[x(i) - x(i+\tau)]^2\right)^{3/2}.$$

Значне відхилення EIx від 0 вказує на асиметрію системи. Якщо $EIx > 0$, розподіл точок на діаграмі Пуанкаре значно зміщений у сторону вище LI. Зворотня ситуація спостерігається для $EIx < 0$. Для $EIx \approx 0$ досліджувані сегменти представляються зворотніми в часі.

11.1.1.5 Індекс площі

Індекс площі (Area index, AIx) [183] визначається як сукупна площа секторів, що сформовані точками над LI поділена на сукупну площу секторів, що відповідають усім точкам на діаграмі Пуанкаре (крім тих, що розташовані точно на LI). Площа сектора, що відповідає певній точці P_i, обчислюється як

$$S_i = 1/2 \times R\theta_i \times r^2,$$

де r — це радіус сектора; $R\theta_i = \theta_{LI} - \theta_i$; θ_{LI} — це фазовий кут, і $\theta_i = \arctan[x(i+\tau)/x(i)]$, що визначає фазовий кут i-ої точки. AIx визначається за формулою:

$$AIx = \sum_{i=1}^{a}|S_i| \Bigg/ \sum_{i=1}^{m}|S_i|.$$

11.1.1.6 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 [184]:

$$SIx = \sum_{i=1}^{a} |R\theta_i| \Big/ \sum_{i=1}^{m} |R\theta_i|.$$

11.1.2 Методи складних мереж

Графи видимості (Visibility graphs, VG) базуються на простому відображенні часових рядів у мережну область, де кожне спостереження є вершиною в складній мережі. Дві вершини i та j пов'язані ребром, якщо для них застосовується наступна умова [178]:

$$x_k < x_j + (x_i - x_j)(t_j - t_k)/(t_j - t_i).$$

де x_k представляє певну перешкоду, якої не має бути, щоб дві вершини можна було зв'язати шляхом.

Матрицю суміжності (A_{ij}) представленого ненаправленого та незваженого VG можна представити як

$$A_{ij}^{VG} = A_{ji}^{VG} = \prod_{k=i+1}^{j-1} H\left[x_k < x_j + (x_i - x_j)(t_j - t_k)/(t_j - t_i)\right],$$

де $H(\cdot)$ — функція Гевісайда.

Граф горизонтальної видимості (Horizontal visibility graph, HVG) є спрощеною версією цього алгоритму [185]. Для досліджуваного часового ряду набори вершин VG і HVG однакові, тоді як набір ребер HVG відображає взаємну горизонтальну видимість двох спостережень x_i та x_j. Тобто, можна побудувати ребро (i, j), якщо $x_k < \min(x_i, x_j)$ для всіх k при $t_i < t_k < t_j$ так що

$$A_{ij}^{VG} = A_{ji}^{VG} = \prod_{k=i+1}^{j-1} H\,(x_i - x_k) H(x_j - x_k).$$

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 мережних показників на основі двох простих характеристик вершин:

1. Оскільки кількість ребер інцидентних вершині i можна визначити як $k_i^r = \sum_j A_{ij}$ для (H)VG ми можемо переписати дану кількісну характеристику для вершини в час t_i відносно її минулих та майбутніх вершин [172,186]:

$$k_i^r = \sum_{j<i} A_{ij} \quad \text{і} \quad k_i^a = \sum_{j>i} A_{ij},$$

де $k_i = k_i^r + k_i^a$, і k_i^r та k_i^a сприймаються як вхідні (минулі) та вихідні (майбутні) вершини.

2. Локальний коефіцієнт кластеризації $C_i = [2/k_i(k_i - 1)] \sum_{j,k} A_{ij} A_{jk} A_{ki}$ інша властивість старшного порядку структурного сусідства вершини i. Для дослідження незворотності ми можемо переписати дані характеристики наступним чином:

$$C_i^r = [2/k_i^r(k_i^r - 1)] \sum_{j<i,k<i} A_{ik}\, A_{jk} A_{ki}$$

і

$$C_i^a = [2/k_i^a(k_i^a - 1)] \sum_{j>i,k>i} A_{ik}\, A_{jk} A_{ki}.$$

Якщо уявити нашу систему зворотною в часі, ми припускаємо, що розподіли ймовірностей прямих і зворотних за часом характеристик повинні бути однаковими. Для незворотних процесів ми очікуємо статистичну нееквівалентність. Ця нееквівалентність буде визначатися через дивергенцію Кульбака-Лейблера [171]:

$$D_{KL}(p||q) = \sum_{i=1}^{N} p\,(x_i) \cdot \log[p(x_i)/q(x_i)],$$

де $p(\cdot)$ відповідатиме розподілу вхідних характеристикам, а $q(\cdot)$ — зворотніх. Крім того, подібність обох величин можна оцінити за допомогою відстані Дженсена-Шеннона [187]:

$$JS(p||q) = \sqrt{[D_{KL}(p||m) + D_{KL}(q||m)/2]},$$

де $m = 0.5 \cdot (p + q)$, а D_{KL} — дивергенція Кульбака-Лейблера.

11.1.3 Незворотність на основі пермутаційних шаблонів

Ідея аналізу **пермутаційних шаблонів** (permutation patterns, PP) спочатку була запропонована Бандтом і Помпе [61] як 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 d_E:

$$\vec{X}(i) = \{x(i), x(i+\tau), \ldots, x(i+[d_E-1]\tau)\},$$

де часова затримка τ відповідає часу розділення між елементами.

Після цього кожен вектор представляється у вигляді порядкового шаблону $\pi = \{r_0, r_1, \ldots, r_{d_E-1}\}$, що має задовільняти наступній умові:

$$x(i+r_0) \leq x(i+r_1) \leq \ldots \leq x\left(i+r_{d_E-1}\right).$$

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 [188].

Різницю між розподілами прямих часових рядів (P^d) та зворотних (P^r) можна оцінити за допомогою дивергенції Кульбака-Лейблера або Дженсена-Шеннона.

11.2 Хід роботи

Підключаємо та встановлюємо необхідні бібліотеки. Для побудови пермутаційних шаблонів нам знадобиться встановити бібліотеку `ordpy` [189], використовуючи наступну команду:

```
!pip install ordpy
```

Для роботи з графами видимості та мірами незворотності на їх основі ми встановимо бібліотеку `ts2vg` про яку ще поговоримо в наступних роботах:

```
!pip install ts2vg

import numpy as np
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 sklearn import preprocessing
from collections import defaultdict, Counter
from ordpy import ordinal_distribution
from tqdm import tqdm
from scipy.spatial import distance
from ts2vg import NaturalVG, HorizontalVG

%matplotlib inline
```

І встановлюємо параметри для побудови графіків:

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                   # розмір фонтів рисунку
'lines.linewidth': 2,                # товщина ліній
'axes.titlesize': 'small',           # розмір титулки над рисунком
'axes.labelsize': size,              # розмір підписів по осям
'legend.fontsize': size,             # розмір легенди
'xtick.labelsize': size,             # розмір розмітки по осі Ox
'ytick.labelsize': size,             # розмір розмітки по осі Ox
"font.family": "Serif",              # сімейство стилів підписів
"font.serif": ["Times New Roman"],   # стиль підпису
'savefig.dpi': 300,                  # якість збережених зображень
'axes.grid': False                   # побудова сітки на самому рисунку
}

plt.rcParams.update(params)          # оновлення стилю згідно налаштувань
```

Далі визначаємо функцію 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 inrange(len(ts))}
```

У подальшому ми представимо індекс Кошти для розрахунку якого визначимо наступну функцію:

```
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 = (DecCosta-IncCosta)/(len(time_ser)-tau)
        Cst.append(C)
    Costa = np.mean(np.abs(Cst))
    return Costa
```

11.2.1 Оголошення функцій для підрахунку показників незворотності

Пермутаційна незворотність:

```
def PermIrrever(time_ser, d_e, tau, delta=1e-10, distance_irr="kullback"):

# створення зворотньої версії ряду
    rev_arr = np.flip(time_ser)

# отримання розподілу порядкових шаблонів для вихідного ряду
    _, dist_dir = ordinal_distribution(time_ser,
                                       dx=d_e,
                                       taux=tau,
                                       return_missing=True)

# отримання розподілу порядкових шаблонів для зворотного ряду
    _, dist_rev = ordinal_distribution(rev_arr,
                                       dx=d_e,
                                       taux=tau,
                                       return_missing=True)

    if distance_irr == "kullback":
        KLD_perm = dist_dir * np.log((dist_dir + delta) / (dist_rev + delta))
        return np.sum(KLD_perm)
    else:
        return distance.jensenshannon(dist_dir + delta, dist_rev + delta)
```

Функція для підрахунку ймовірностей:

```
def calc_prob_dist(p, q):

    p_cnt, q_cnt = dict(Counter(p)), dict(Counter(q))
    p_sum = sum(p_cnt.values())
    p_dist = {k: v / p_sum for k, v in p_cnt.items()}
    q_sum = sum(q_cnt.values())
    q_cnt = {k: v / q_sum for k, v in q_cnt.items()}
    q_dist = defaultdict(lambda: 0)
    q_dist.update(q_cnt)

    for k in p_dist.keys():
        if k not in q_dist.keys():
            q_dist[k] = 0.0

    for k in q_dist.keys():
```

```
         if k not in p_dist.keys():
            p_dist[k] = 0.0

     return p_dist, q_dist
```

Дивергеція Кульбака-Лейблера:

```
def KLD(p_dist, q_dist, delta=1e-10):

    div_list = [
        p_proba * np.log((p_proba + delta) / (q_dist[k] + delta)) for k, p_proba
in p_dist.items()
    ]

    kld = np.sum(np.array(div_list))

    return kld
```

Дивергенція Дженсена-Шеннона:

```
def JS(p_dist, q_dist, delta=1e-10):

    m_dist = {k: 0.5*(p_dist[k]+q_dist[k]) for k in p_dist.keys()}

    js = 0.5*KLD(p_dist, m_dist) + 0.5*KLD(q_dist, m_dist)

    return js
```

Графо-динамічна незворотність:

```
def GraphIrrever(fragm,
                 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)
     elif graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
     else:
        g = recurrence_net(fragm,
                           rec_thr=eps_rec*np.abs(np.std(fragm)),
                           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)
```

```
# знаходимо розподіл імовірностей
    ret_deg_probs, adv_deg_probs = calc_prob_dist(ret_deg, adv_deg)
    ret_clust_probs, adv_clust_probs = calc_prob_dist(ret_clust, adv_clust)

# розраховуємо асиметрію (незворотність) за допомогою Кульбака-Лейблера
    if distance_irr == "kullback":
        distance_deg = KLD(ret_deg_probs, adv_deg_probs, delta)
        distance_clust = KLD(ret_clust_probs, adv_clust_probs, delta)

# розраховуємо асиметрію (незворотність) за допомогою Йенсена-Шеннона
    if distance_irr == "shannon":
        distance_deg = JS(ret_deg_probs, adv_deg_probs, delta)
        distance_clust = JS(ret_clust_probs, adv_clust_probs, delta)

    return distance_deg, distance_clust
```

11.2.1.1 Функції для отримання ступеня вершини та локальної кластеризації

Процедура знаходження ступеня вершини та локальної кластеризації кожної вершини є доволі громіздкою. Для прискорення розрахунків відповідних процедур скористаємось бібліотекою `numba`. Numba — 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
```

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
```

Розглянемо можливість використання всіх згаданих показників у якості індикаторів або індикаторів-передвісників кризових явищ. Для прикладу завантажимо часовий ряд фондового індексу Russell 2000 за період із 1 грудня 1988 року по 1 листопада 2023 року:

```
symbol = '^RUT'                         # символ індексу
start = '1988-12-01'                    # початок відліку
end = '2023-11-01'                      # кінець відліку
data = yf.download(symbol)              # вивантажуємо дані
time_ser = data['Adj Close'].copy()     # зберігаємо саме ціни закриття

xlabel = 'time, days'                   # підпис по вісі Ox
ylabel = symbol                         # підпис по вісі Oy

date_in_num = mdates.date2num(time_ser.index)
```

```
np.savetxt(f'{symbol}_initial_time_series.txt', time_ser.values)
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того, з яким рядом ми працюємо

```
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 до окремої змінної

xlabel = r'$\varepsilon$'            # підпис по вісі Ox
ylabel = symbol                      # підпис по вісі Oy

date_in_num = time_ser.index
```

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 Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

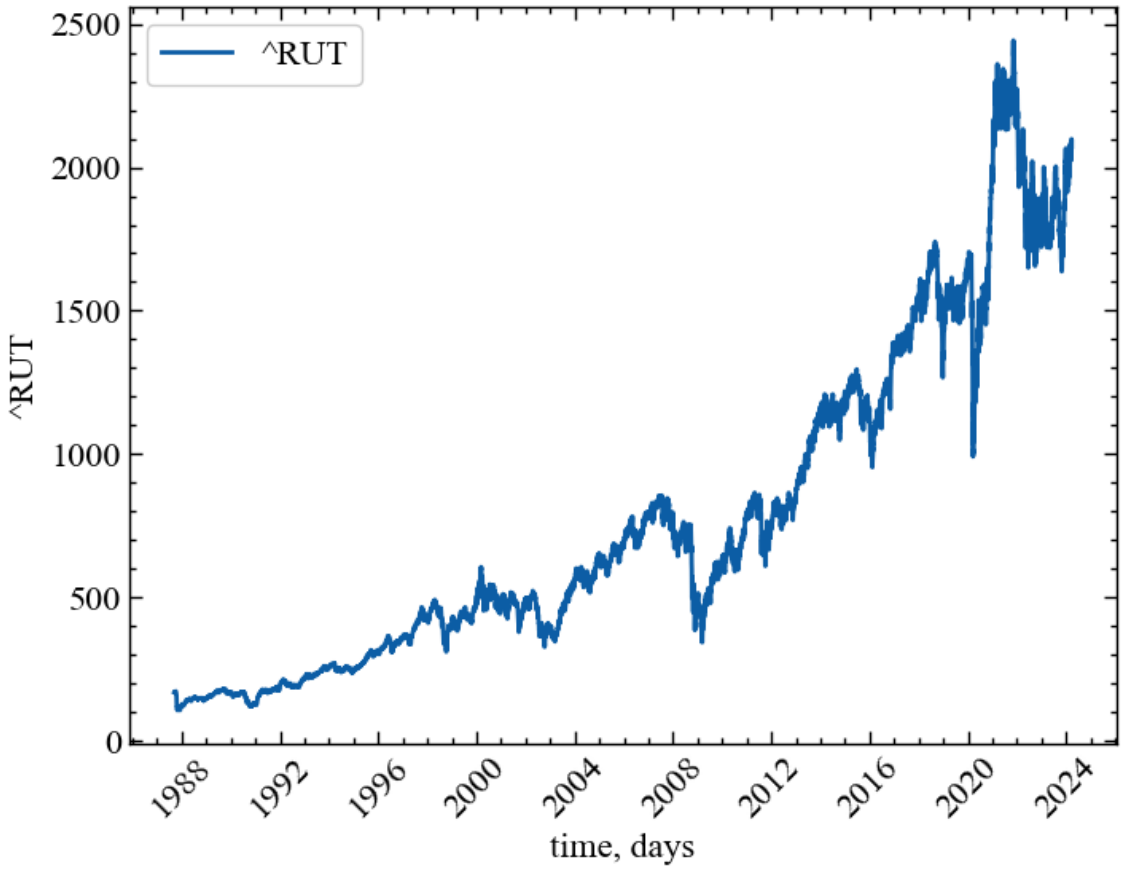


Рис. 11.1: Динаміка щоденних змін індексу Russell 2000

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. 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
```

11.2.2 Виводимо діаграму Пуанкаре та розраховуємо міри на її основі

Розглянемо діаграму Пуанкаре для фрагмента прибутковостей часового ряду:

```
for_puank = time_ser.copy()

tau_assym = 1 # часова затримка для діаграми Пуанкаре

ret_type = 4 # тип ряду:
# 1 - вихідний,
# 2 - детрендований
# 3 - прибутковості звичайні,
# 4 - стандартизовані прибутковості,
# 5 - абсолютні значення (волатильності)
# 6 - стандартизований вихідний часовий ряд

idx_beg = 3000 # кінцевий відлік
idx_end = 5000 # початковий відлік

fragm = for_puank[idx_beg:idx_end]  # виокремлюємо фрагмент ряду

for_puank = transformation(fragm, 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}_{idx_beg}_{idx_end}.jpg",
bbox_inches="tight")
plt.show();
```

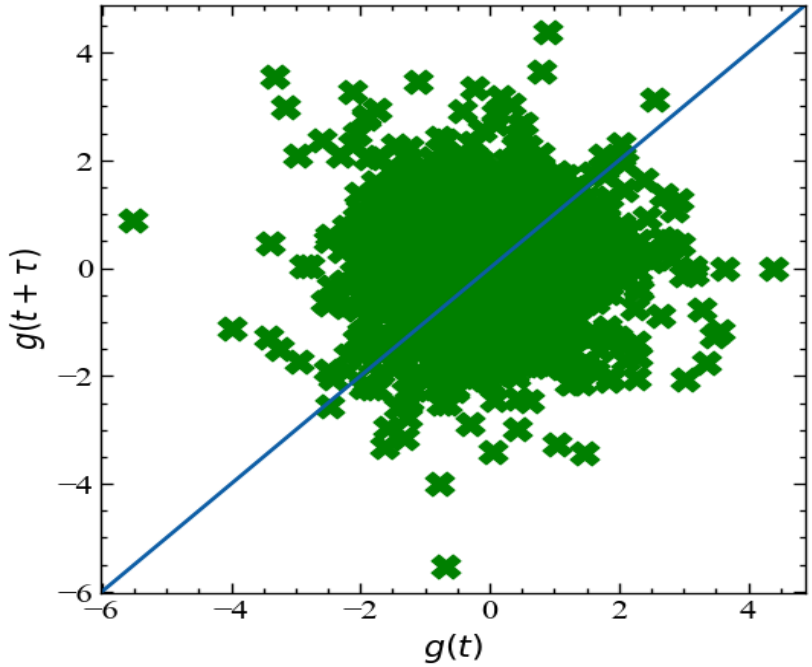


Рис. 11. 2.: Діаграма Пуанкаре для фрагмента індексу Russell 2000

Виходячи з даної діаграми, можна припустити, що для прибутковостей індексу Russell 2000 спостерігається асиметрія у сторону негативних флуктуацій ряду.

11.2.3 Віконна процедура

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', labelrotation=45, **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();
```

Визначаємо параметри та оголошуємо масиви для збереження результатів:

```
window = 500 # розмір ковзного вікна
tstep = 1 # часовий крок

ret_type = 1 # тип ряду:
# 1 - вихідний,
# 2 - детрендований
# 3 - прибутковості звичайні,
# 4 - стандартизовані прибутковості,
# 5 - абсолютні значення (волатильності)
# 6 - стандартизований вихідний часовий ряд

# 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)

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)
```

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)
```

11.2.4 Візуалізація показників на основі діаграми Пуанкаре

11.2.4.1 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")
```

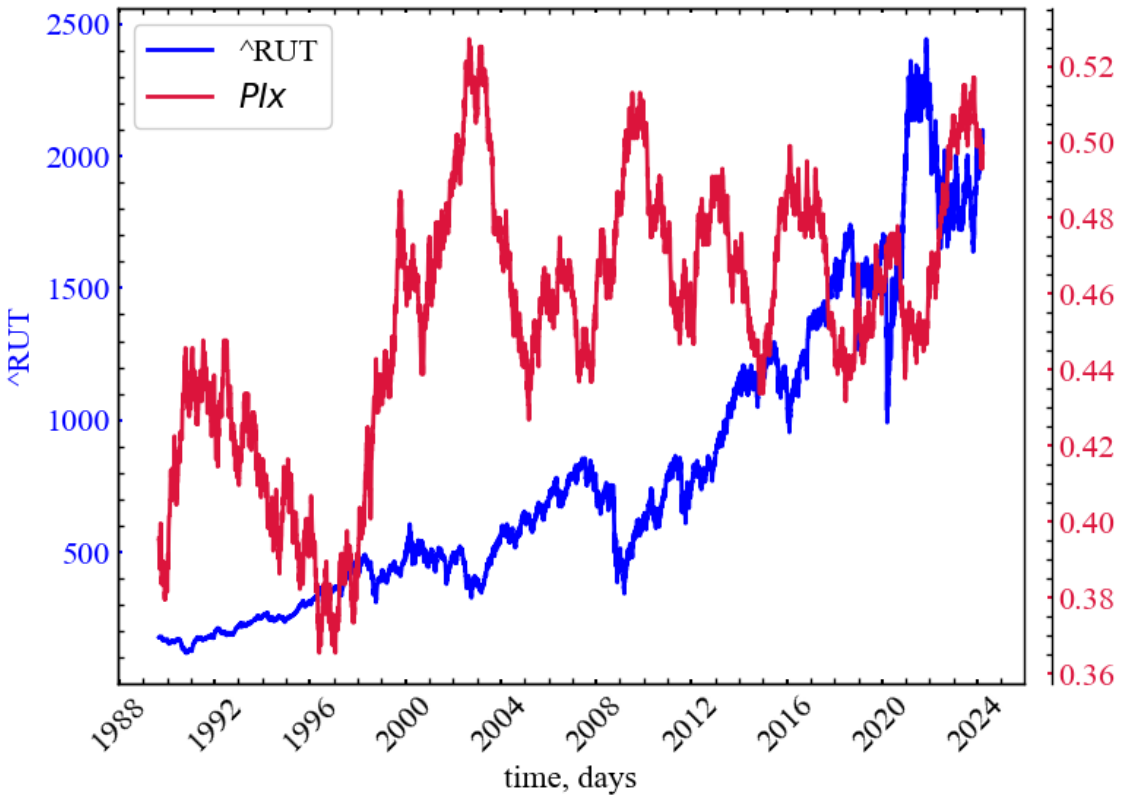


Рис. 11.3: Динаміка індексу Russell 2000 та індексу Порти

Як можна бачити з Рис. 11.3, індекс Порти спадає у передкризові періоди, що вказує на переважне зростання частки позитивних прибутковостей у межах вікна в 500 днів. У кризові періоди індекс Порти зростає, що вказує на домінацію негативного тренду (точок нижче головної діагоналі) в цінових флуктуаціях Russell 2000. Видно, що PIx закономірно спадає перед кризами 1997, 2008, 2011, 2015, 2020 років. Тобто, даний показник можна використовувати в якості індикатора-передвісника крахових подій на фондовому ринку.

11.2.4.2 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")
```

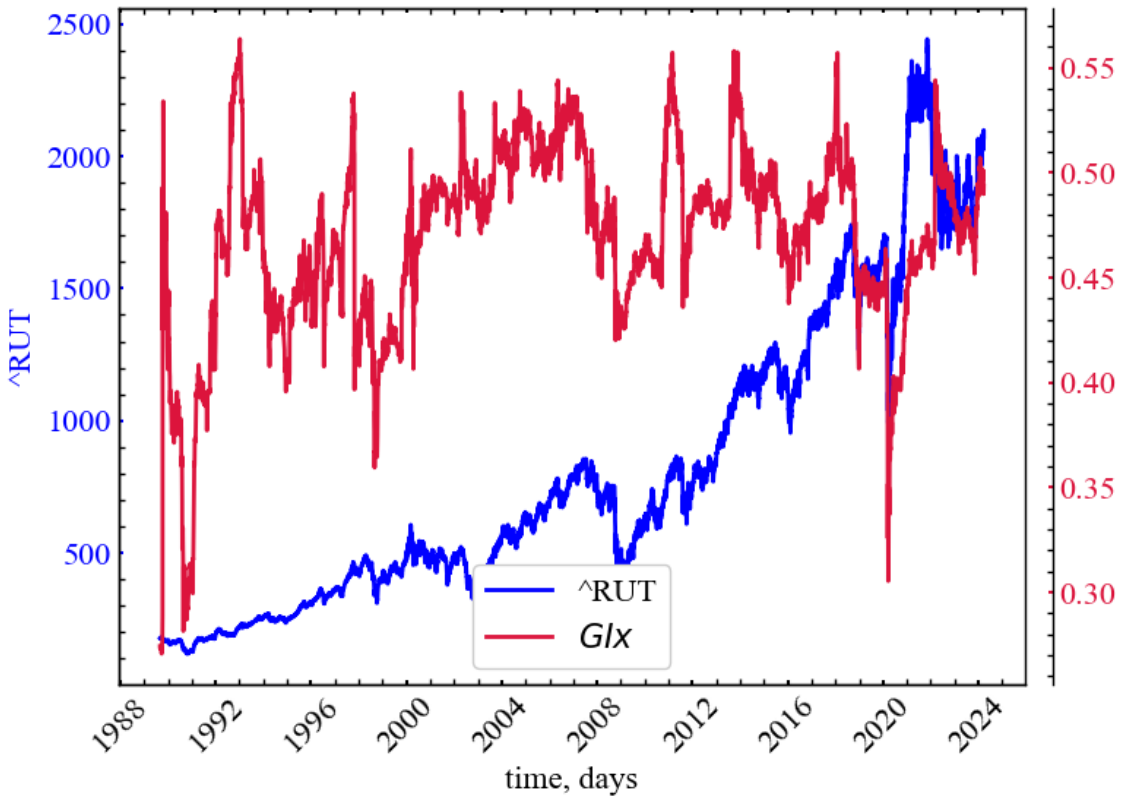


Рис. 11.4: Динаміка індексу Russell 2000 та індексу Гузіка

На рисунку Рис. 11.4 спостерігається спад індексу Гузіка в кризові та передкризові періоди. Це говорить про те, що сумарна частка точок, яка знаходились вище LI стала меншою. Також варто зазначити, що ці точки в передкризові періоди стають усе ближче до головної діагоналі, тому й сума квадратів відстаней точок вище LI поступово стає меншою. Подібного роду поведінку можна використовувати для передчасної ідентифікації крахових подій на фондовому ринку.

11.2.4.3 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")
```

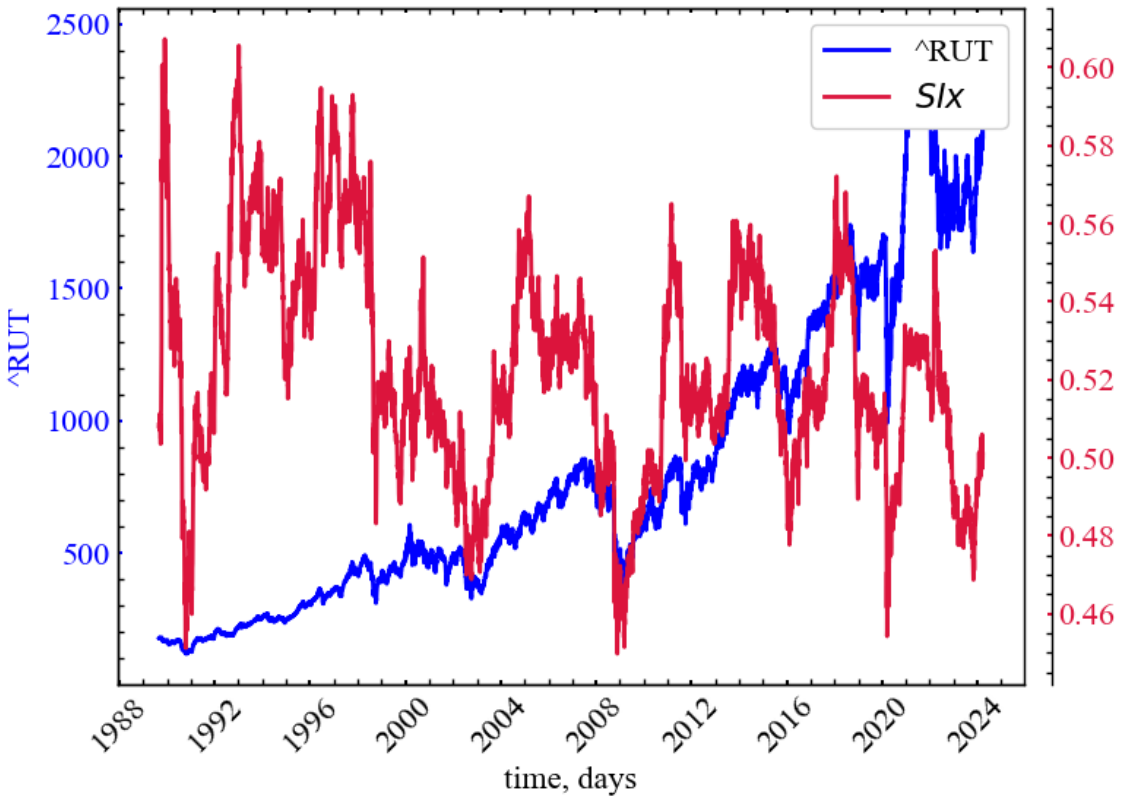


Рис. 11.5: Динаміка індексу Russell 2000 та індексу кута нахилу

На Рис. 11.5 у закономірний спосіб спостерігається спад індексу кута нахилу в передкрахові періоди Russell 2000. Це говорить про спад сумарної частки точок над LI, які формували певних кут нахилу. У перекризові та кризові періоди вони починають “лягати” на LI, що робить кут нахилу деяких із цих точок близьким до нуля. Подібну закономірність також можна використовувати для передбачення кризових подій.

11.2.4.4 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")
```

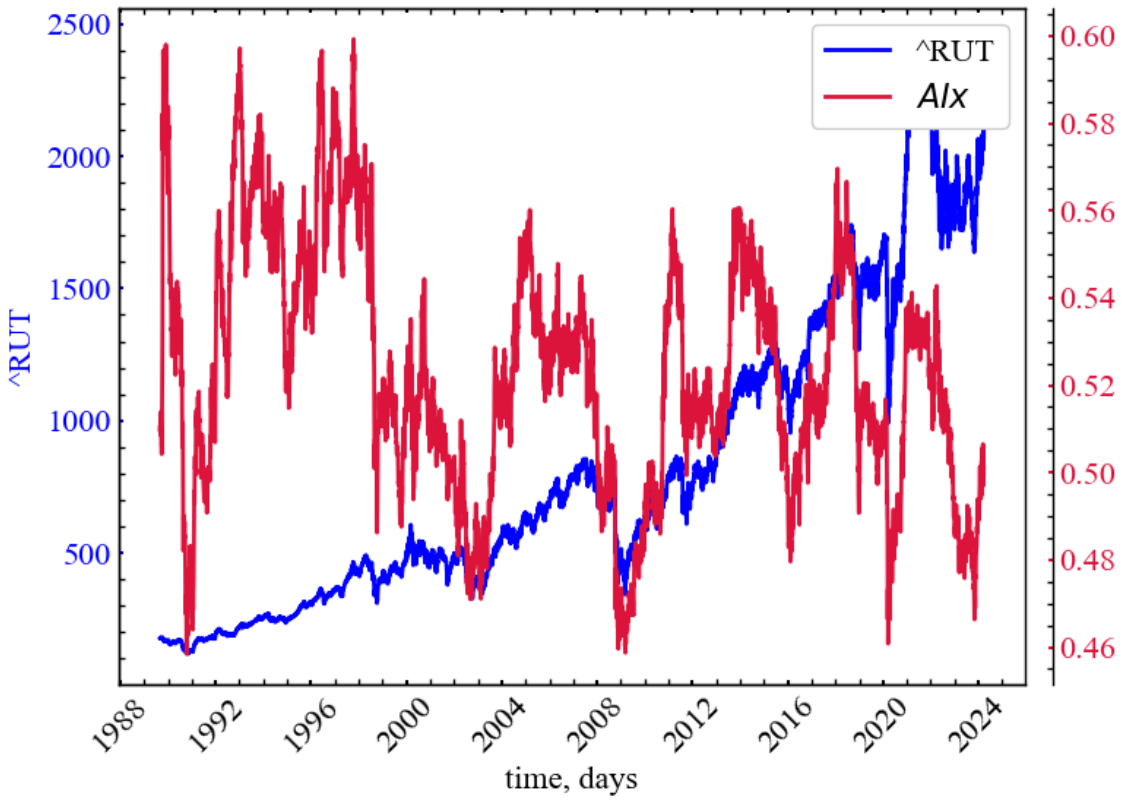


Рис. 11.6: Динаміка індексу Russell 2000 та індексу площі секторів

Рис. 11.6 демонструє характерний спад індексу площі в передкрахові періоди Russell 2000. Це говорить про спад сумарної частки точок над LI, які формували площі. У перекризові та кризові періоди вони починають “лягати” на LI, що робить їх результуючу площу близькою до нуля. Як можна бачити, площі секторів точок над LI закономірно спадають у передкрахові періоди найбільш ключових криз на фондовому ринку.

11.2.4.5 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")
```

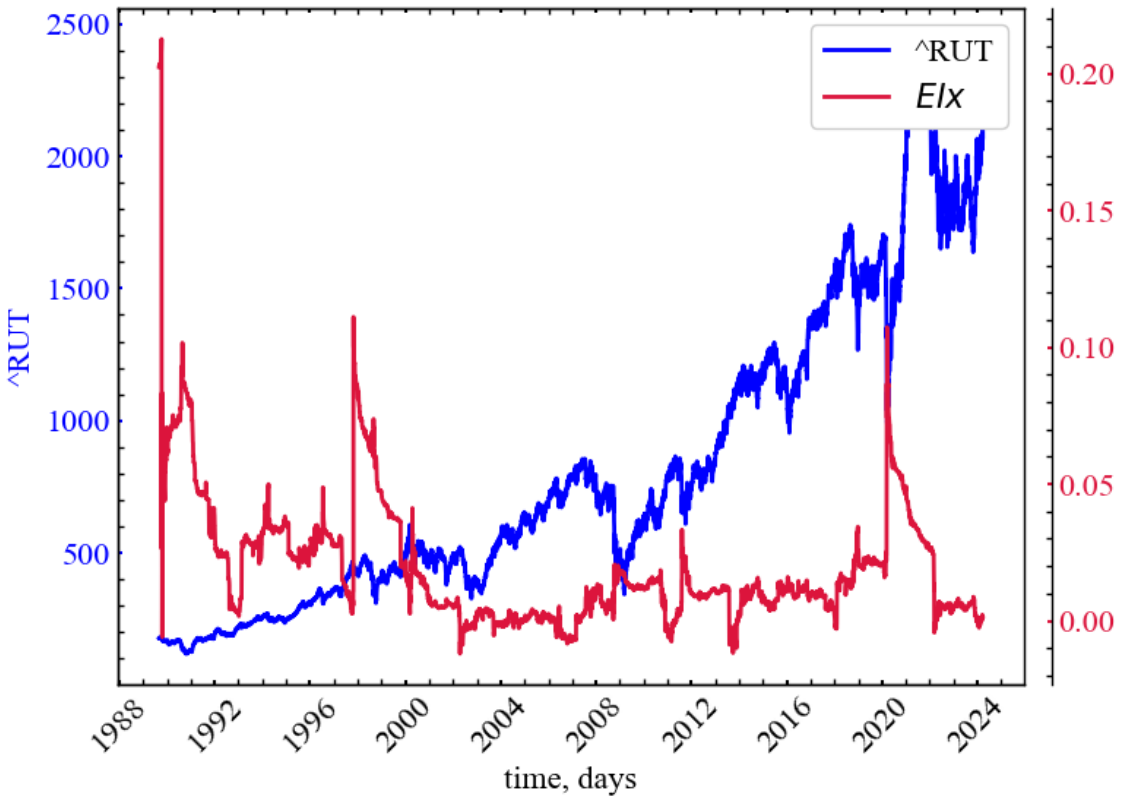


Рис. 11.7: Динаміка індексу Russell 2000 та індексу Ейлера

Рис. 11.7 показує, що індекс Ейлера зростає під час кризових моментів. Це говорить про зростання різниці між наступними значеннями та попередніми. Даний показник можна пробувати використовувати в якості індикатора, але не передвісника крахових подій.

11.2.4.6 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")
```

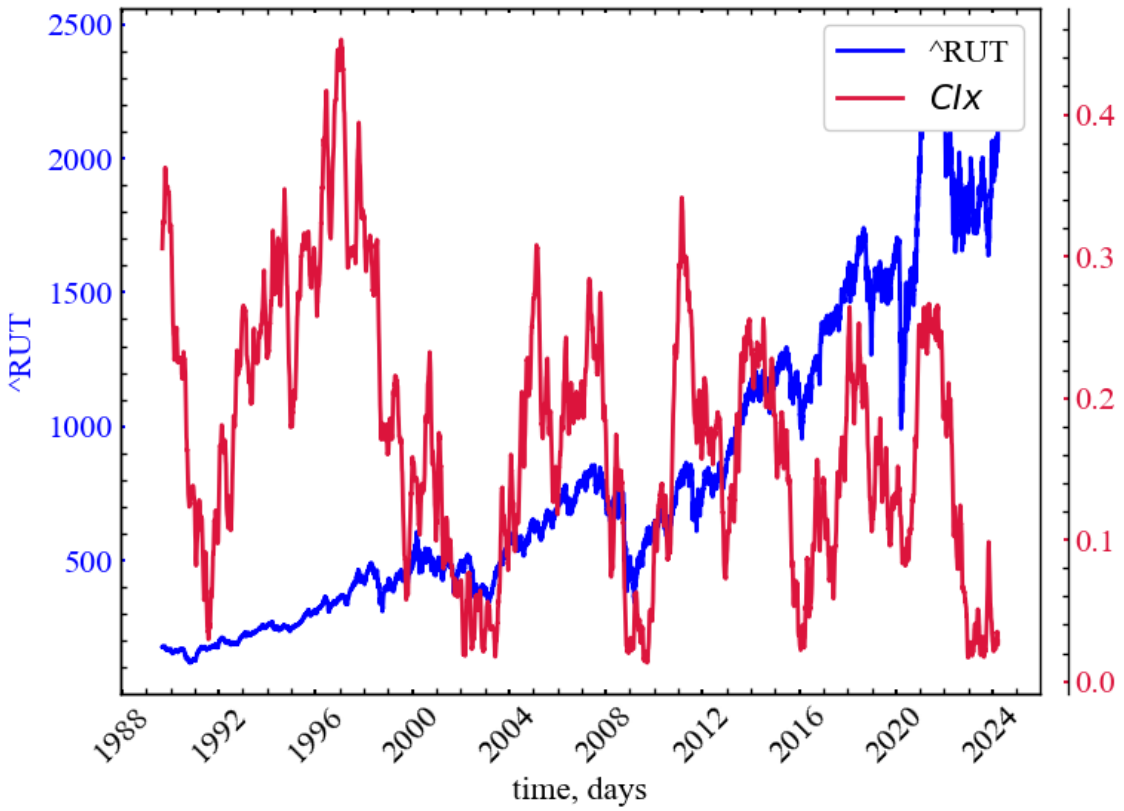


Рис. 11.8: Динаміка індексу Russell 2000 та індексу Кошти

На Рис. 11.8 можна спостерігати зростання CIx у передкризові періоди, що вказує на зростання асиметрії між декрементами та інкрементами в динаміці досліджуваного індексу. Подібну поведінку можна розглядати в якості індикатора незворотності системи до входження у стан краху.

11.2.5 Побудова графу досліджуваного ряду

```
graph_type = 'classic'  # тип графу: classic, horizontal, recurrent

# 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'.


index_begin = 2000 # початковий індекс для графу
index_end = 4000   # кінцевий індекс для графу

ret_type = 1       # вид ряду
```

```
# 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)],
}

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)
    pos1 = 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)
    pos1 = 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)

    pos1 = node_positions_recurrence_net(for_graph_plot[index_begin:index_end],
date)
    nxg = g
```

Виводимо зв’язки видимості:

```
fig, ax = plt.subplots(1, 2, figsize=(13, 8))

nx.draw_networkx(nxg, ax=ax[0], pos=pos1, **graph_plot_options)
ax[0].tick_params(bottom=True, labelbottom=True)
ax[0].plot(time_ser.index[index_begin:index_end],
for_graph_plot[index_begin:index_end], label=fr"{ylabel}")
ax[0].set_title(f'Зв\'язки видимості для {ylabel}', pad=10)
ax[0].set_xlabel(xlabel)
ax[0].set_ylabel(f"{ylabel}")
ax[0].legend(loc='upper right')
ax[0].tick_params(axis='x', labelrotation=45)

ax[1].set_title(f'Графове представлення для {symbol}', pad=10)

# визначаємо позицію вузлів на графі
pos2 = nx.spring_layout(nxg, k=0.15, iterations=100)
```

```
# розраховуємо ступеневу центральність
degCent = nx.degree_centrality(nxg)

# створити список розмірів вершин на основі ступеневої центральності
node_sizes = [v*100 for v in degCent.values()]

# кольори вузлів на основі їх ступеневої центральності
node_colors = [v for v in degCent.values()]

# будуємо граф
nx.draw_networkx(nxg,
                 ax=ax[1],
                 pos=pos2,
                 node_size=node_sizes,
                 node_color=node_colors,
                 with_labels=False,
                 cmap=plt.get_cmap('plasma'))

# присвоюємо мінімальне та максимальне значення
# ступеневої центральності для побудови теплової шкали
vmin = np.asarray(list(degCent.values())).min()
vmax = np.asarray(list(degCent.values())).max()

sm = plt.cm.ScalarMappable(cmap=plt.get_cmap('plasma'),
                           norm=plt.Normalize(vmin=vmin, vmax=vmax))
cb = plt.colorbar(sm, ax=ax[1])
cb.set_label('Ступенева центральність')

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", dpi=1000)
```

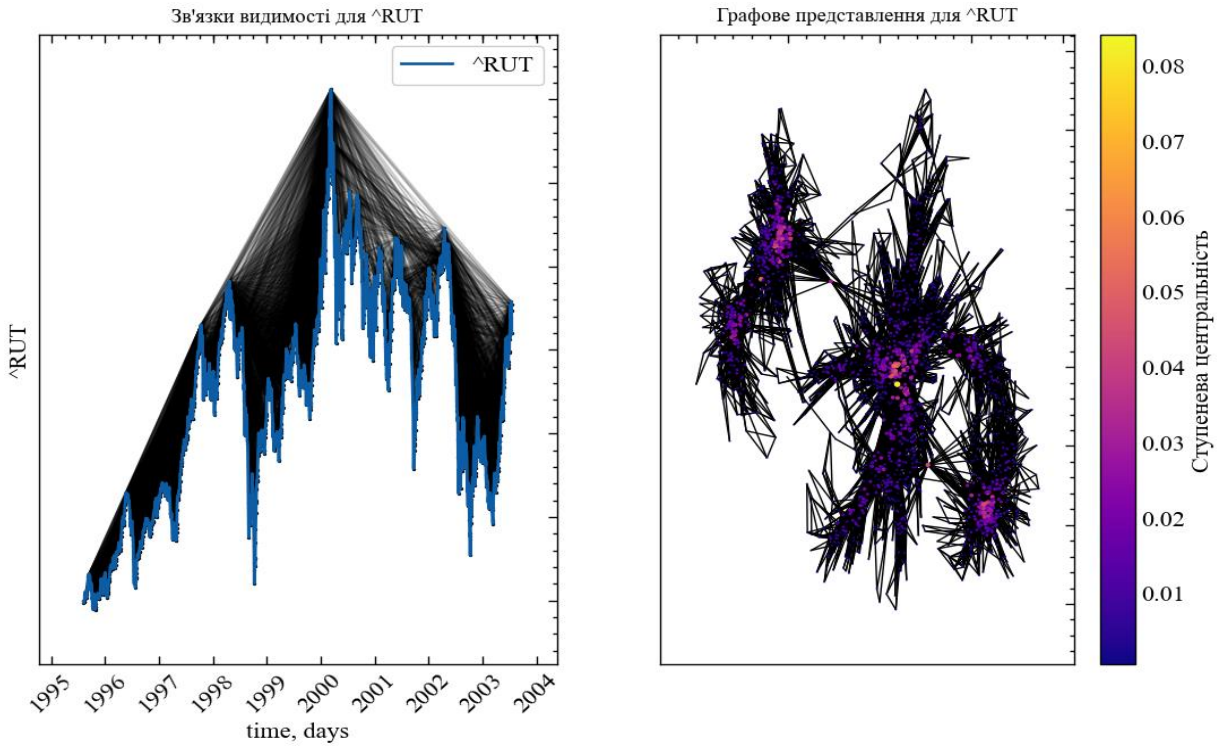


Рис. 11.9: Графік зв’язків видимості на основі природного VG напередодні крахів 2001-го року на індексі Russell 2000 та графове представлення цього фрагмента

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.

11.2.6 Побудова показників незворотності на основі пермутаційних шаблонів та графів

Ініціалізуємо масиви для збереження результатів розрахунків:

```
window = 500 # ширина вікна
tstep = 1 # часовий крок

ret_type = 1 # вид ряду:
# 1 - вихідний
# 2 - детрендований (різниця між теп. значенням та попереднім)
# 3 - прибутковості
# 4 - стандартизовані прибутковості
# 5 - абсолютні значення (волатильності)
# 6 - стандартизований 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

length = len(time_ser.values)  # довжина самого ряду

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)
```

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)
```

11.2.7 Візуалізація показників на основі графів та пермутаційних шаблонів

11.2.7.1 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")
```

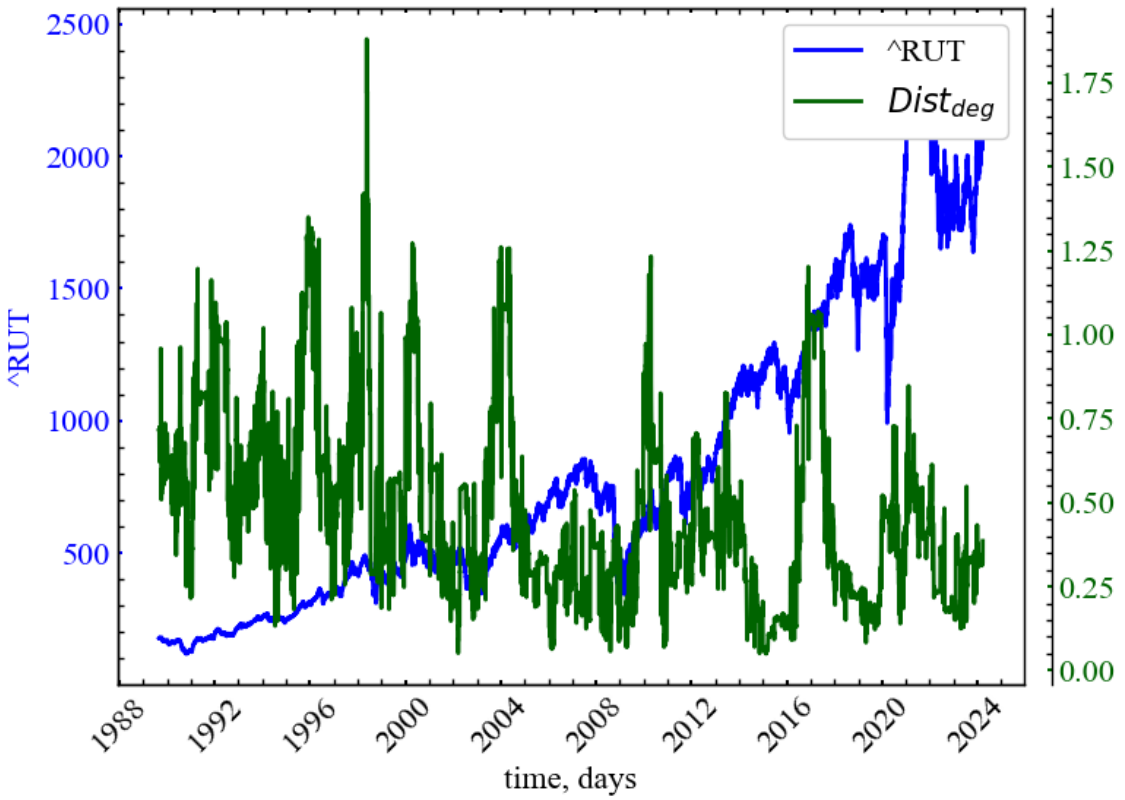


Рис. 11.10: Динаміка індексу Russell 2000 та показника незворотності на основі ступеня вершини

Показник незворотність на основі ступенів вершини починає зростати в передкризові періоди, вказуючи на зростання асиметрії вхідних та вихідних ступенів вершини в предкризові періоди. У момент самої кризи можна спостерігати зростання симетрії цих характеристик.

11.2.7.2 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="darkred")
```

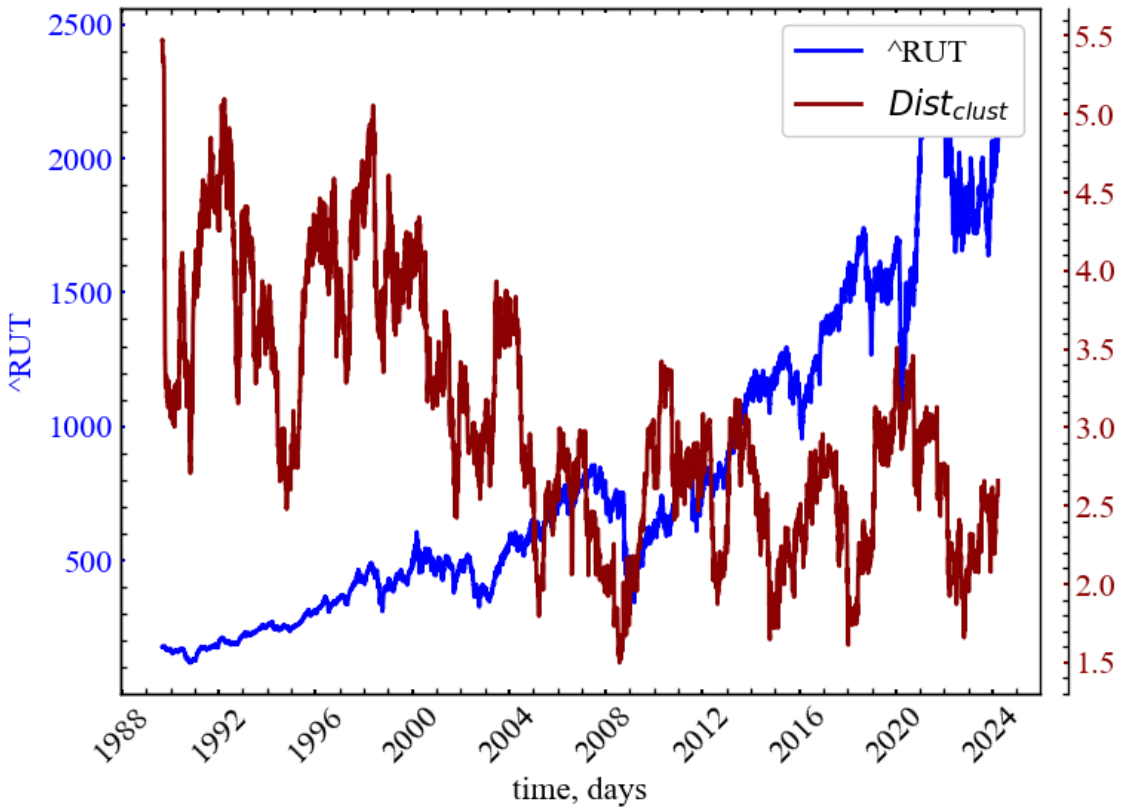


Рис. 11.11: Динаміка індексу Russell 2000 та показника незворотності на основі локальної кластеризації

На Рис. 11.11 видно, що в передкризові та кризові періоди незворотність системи зростає, що відображається асиметрією розподілом локальних коефіцієнтів кластеризації вхідних і вихідних вершин.

11.2.7.3 Ступінь незворотності на основі пермутаційних шаблонів

```
measure_label = r"$Dist_{perm}$"
file_name = f"Perm_symbol={symbol}_wind={window}_ \
            step={tstep}_ret_type={ret_type}\
            d_e={d_e_perm}_tau={tau_perm}\
            _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="black")
```

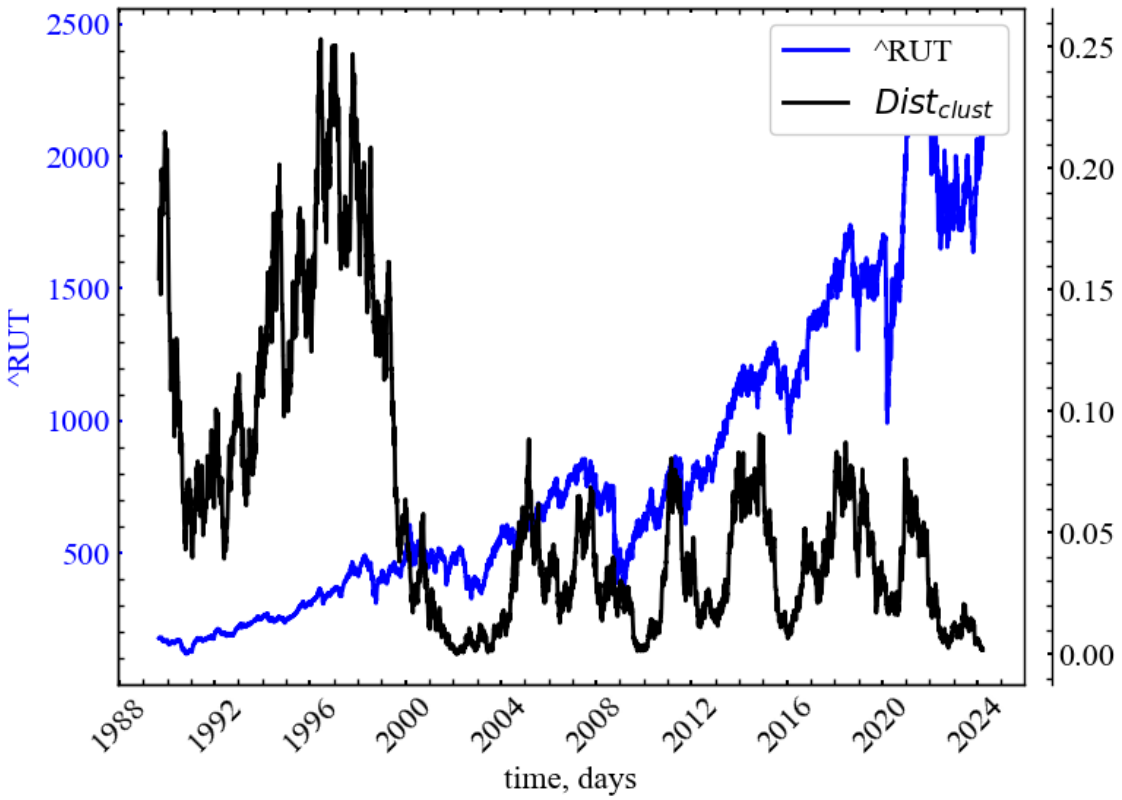


Рис. 11.12: Динаміка індексу Russell 2000 та показника незворотності на основі пермутаційних шаблонів

Рис. 11.12 демонструє зростання $Dist_{perm}$ у передкризові періоди, що слугує індикатором біфуркації ринку та початку незворотніх трансформацій. Найвищий ступінь незворотності спостерігався саме напередодні кризи 1997 року. Пермутаційна незворотність закономірно зростає напередодні майже всіх фондових криз, але подальша динаміка $Dist_{perm}$ не перевищує зростання напередодні початку нового століття.

11.3 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 незворотності (асиметрії) системи в дані періоди 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 передкризові періоди або прямо в момент кризи, вказуючи на стартовий період хаосу. Найгірше серед них себе поводить $Dist_{deg}$. Найращим чином $Dist_{clust}$ та $Dist_{perm}$. Їх і варто використовувати в якості індикаторів-передвісників крахів.

11.4 Завдання для самостійної роботи

1. Виберіть часовий ряд згідно вашого варіанту
2. Виділіть на вашому ряді стани стабільності та криз і проаналізуйте їх із використанням індексів асиметрії та незворотності
3. Проведіть варіацію часового вікна та кроку. Які висновки можна зробити?

12. Лабораторна робота № 12

Тема. 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, спираючись 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 може визначати 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 вибірок. Аналітики та трейдери завжди будуть дивуватись новим екстремальним подіям, що, як правило, рідко бувають позитивними.

12.1 Теоретичні відомості

12.1.1 Імпортуємо необхідні бібліотеки

> ⓘ Увага
>
> Перш за все, для подальшої роботи, нам необхідно буде встановити бібліотеку `pylevy`. Встановити її можна з відповідного GitHub репозиторію (https://github.com/josemiotto/pylevy/tree/master). Ми у свою чергу завантажимо її напряму через команду `pip install` слідуючим чином:

```
!pip install git+https://github.com/josemiotto/pylevy.git
```

Після встановлення необхідного модулю можна приступати до подальшої роботи.

```
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
```

12.1.2 Виконуємо налаштування рисунків

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
'figure.figsize': (8, 6),           # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                  # розмір фонтів рисунку
'lines.linewidth': 2,               # товщина ліній
'axes.titlesize': 'small',          # розмір титулки над рисунком
'axes.labelsize': size,             # розмір підписів по осям
'legend.fontsize': size,            # розмір легенди
'xtick.labelsize': size,            # розмір розмітки по осі Ox
'ytick.labelsize': size,            # розмір розмітки по осі Ox
"font.family": "Serif",             # сімейство стилів підписів
"font.serif": ["Times New Roman"],  # стиль підпису
'savefig.dpi': 300,                 # якість збережених зображень
'axes.grid': False                  # побудова сітки на самому рисунку
}

plt.rcParams.update(params)         # оновлення стилю згідно налаштувань
```

12.1.2 Розподіл Леві

(Альфа, α)-стабільний розподіл (α-stable distribution) є узагальненням розподілу Гауса, що враховує “важкі 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 [192].

Раніше ця тема привертала лише помірну увагу провідних експертів. Натхненням для Леві стало бажання узагальнити відому центральну граничну

теорему, згідно з якою будь-який розподіл імовірностей з кінцевою дисперсією збігається до розподілу Гауса.

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.

12.1.2.1 Інваріантність при додавані

Випадкова величина X підпорядковується стабільному розподілу $P(x) = \text{Prob}\{X \leq x\}$*, якщо для будь-якого* $n \geq 2$ *існують додатнє значення* c_n *та дійсне значення* d_n *такі, що*

$$X_1 + X_2 + \ldots + X_n \stackrel{d}{=} c_n X + d_n,$$

де $X_1, X_2, \ldots, X_n$ *характеризуються як незалежні, ідентично розподілені випадкові величини.* Також $\stackrel{d}{=}$ позначає *рівність розподілів*, тобто, випадкові величини з обох сторін мають однаковий розподіл імовірностей.

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 прямує до величини такого самого розподілу [193–195]. У цьому випадку результуюча випадкова величина (розподіл) може відрізнятися від попередніх величин характерним масштабом (c_n) та зміщенням (d_n). Якщо $d_n = 0$, розподіл називається *строго стабільним*.

Відомо, що нормована константа c_n має вид

$$c_n = n^{1/\alpha} \text{ при } 0 < \alpha \leq 2.$$

Параметр α є **характеристичною експонентою** або **індексом стабільності** розподілу.

Попередня теорема має альтернативну версію, що включає в суму лише дві випадкові величини. *Випадкова величина X підпорядковується стабільному розподілу, якщо для будь-яких позитивних значень A та B існує позитивне число C та дійсне число D такі, що*

$$AX_1 + BX_2 \stackrel{d}{=} CX + D,$$

де X_1 *та* X_2 *незалежні копії* X. Тоді існує значення $\alpha \in (0,2]$ при яких значення C задовільняє рівність $C^\alpha = A^\alpha + B^\alpha$.

Для строго стабільних розподілів $D = 0$. Це означає, що всі лінійні комбінації випадкових незалежних, ідентично розподілених величин, що підкоряються строго стабільному розподілу, результують у випадкову величину з одним і тим же типом розподілу.

Стабільний розподіл вважається *симетричним*, якщо величина $-X$ має такий самий тип розподілу. *Симетричний* стабільний розподіл обов'язково *строго стабільний*.

Оскільки аналітичний вираз функції щільності ймовірностей для стабільного розподілу невідомий, за винятком кількох членів стабільного сімейства, більшість традиційних методів математичної статистики не можуть бути використані. Відповідними вийнятками є

1. **Гаусовий розподіл** $S_2(0, \mu, \sigma) = \mathcal{N}(\mu, 2\sigma^2)$. Гаусовий розподіл є спеціальним випадком стабільного розподілу при $\alpha = 2$ так що $\mathcal{N}(\mu, \sigma) = S\left(2,0, \mu, \sigma/\sqrt{2}\right)$, де μ позначає середнє значення нормального розподілу, а σ — це стандартне відхилення. Функція щільності ймовірностей має вид

$$\left(\sigma\sqrt{2\pi}\right)^{-1}\exp[-(x-\mu)^2/2\sigma^2].$$

2. **Розподіл Коші**. Розподіл Коші — це ще одне представлення стабільного розподілу при $\alpha = 1$ та $\beta = 0$ такими, що $Cauchy(\delta, \gamma) = S_1(1,0, \gamma, \delta)$, де γ — це параметр масштабування, а δ — це параметр зсуву розподілу Коші. Функція щільності ймовірностей представлена як

$$\gamma/\pi[(x-\delta)^2 + \gamma^2],\ -\infty < x < \infty.$$

3. **Розподіл Леві** також є вийнятком із класу стабільних розподілів, де $\alpha = 0.5$ і $\beta = 1$. Іншими словами, $Levy(\delta, \gamma) = S_{1/2}(0.5, 1, \gamma, \delta)$. Функція щільності ймовірностей має вид

$$\sqrt{\gamma/2\pi}(x-\delta)^{-3/2}\exp[-\gamma/2(x-\delta)]\,,\ \delta < x < \infty.$$

Якщо $X \sim S_{1/2}(0.5,1, \gamma, \delta)$, тоді для $x > 0$

$$P(X \le x) = 2\left[1 - \phi\left(\sqrt{\gamma/x}\right)\right],$$

де ϕ позначає кумулятивну функцію нормального розподілу.

12.1.2.2 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 X має область збіжності, тобто якщо існує послідовність незалежних, ідентично розподілених випадкових величин $Y_1, Y_2, \ldots$ і послідовності позитивних чисел γ_n і дійсних чисел δ_n таких, що

$$(Y_1 + Y_2 + \ldots + Y_n)/\gamma_n + \delta_n \overset{d}{\Rightarrow} X.$$

Коли X це гаусова випадкова величина, а Y_i є незалежними, ідентично розподіленими випадковими величинами з визначенною дисперсією, тоді рівняння вище є твердженням звичайної **центральної граничної теореми**. Область збіжності X вважається *нормальною* коли $\gamma_n = n^{1/\alpha}$.

12.1.2.3 Канонічні представлення характеристичної функції

Чотири параметри використовуються для опису випадкової величини, що слідує за стабільним розподілом: $X \sim S(\alpha, \beta, \mu, \gamma)$. Параметр $\alpha \in (0, 2]$ — це той, який нас найбільше зацікавить. Цей параметр визначає товщину хвостів. Параметр $\beta \in [-1, 1]$ є параметром асиметрії. Останні два параметри позначають розташування $(\mu \in \Re)$ і масштаб $(\gamma > 0)$ розподілу. Альфа-стабільний розподіл немає жодного аналітичного виразу для щільності ймовірності X, але ми можемо охарактеризувати його характеристичною функцією [196–198]:

$$\begin{aligned} \phi(t) &= E[\exp(itX)] \\ &= \begin{cases} \exp(i\mu t - \gamma^\alpha |t|^\alpha [1 - i\beta \operatorname{sign}(t) \tan(\pi\alpha/2)]) & \text{при } \alpha \neq 1, \\ \exp(i\mu t - \gamma|t|[1 + i\beta \operatorname{sign}(t)(2/\pi)\log|t|]) & \text{при } \alpha = 1. \end{cases} \end{aligned}$$

Ми могли б використовувати перетворення Фур'є, щоб отримати функцію щільності розподілу ймовірностей з характеристичної функції [199]:

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \phi(t) \exp(-itX)\, dt.$$

Але наведена вище параметризація не є повністю задовільною, оскільки функція щільності розподілу ймовірностей не є неперервною, зокрема, при $\alpha = 1$. Дійсно, коли $\beta > 0$, щільність розподілу зміщується вправо, коли $\alpha < 1$ і

вліво, коли $\alpha > 1$, зі зсувом в сторону $+\infty$ (відповідно $-\infty$), коли α 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 є 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 дещо інші. Нолан запропонував використовувати параметризацію Золотарьова [200], яку також часто позначають як S^0. Характеристична функція, що відповідає $X \sim S^0(\alpha, \beta, \mu_0, \gamma)$, дорівнює:

$$\phi(t) = E[\exp(itX)] = \begin{cases} \exp(i\mu_0 t - \gamma^\alpha |t|^\alpha [1 + i\beta \operatorname{sign}(t) \tan(\pi\alpha/2)\,(\gamma^{1-\alpha}|t|^{1-\alpha} - 1)]) & \text{при } \alpha \neq 1, \\ \exp(i\mu_0 t - \gamma|t|[1 + i\beta \operatorname{sign}(t)(2/\pi)(\log|t| + \log\gamma)]) & \text{при } \alpha = 1. \end{cases}$$

Ця альтернативна параметризація недалека від зазначеної напочатку. Єдина відмінність стосується параметра μ, який у даній параметризації коригує зсув для значень α близьких до 1:

$$\mu_0 = \begin{cases} \mu + \beta\gamma \tan(\pi\alpha/2) & \text{при } \alpha \neq 1, \\ \mu + \beta(2/\pi)\gamma \log\gamma & \text{при } \alpha = 1. \end{cases}$$

12.1.2.4 Метод розрахунку параметрів α-стабільного розподілу

Числові методи, такі як метод заснований на квантилях [201–203], і метод оцінки максимальної правдоподібності [204] були розроблені на виклик відсутності аналітичних рішень. Припустимо, що $\mathrm{X} = (X_1, \ldots, X_T)$ вектор, що складається з T незалежних ідентично розподілених випадкових величин із розподілу Парето, і також $x \sim S_\alpha(\alpha, \beta, \delta, \gamma)$. Визначивши $\theta = (\alpha, \beta, \delta, \gamma)$, Митник, Доганоглу та Ченайо використали алогритм максимальної правдоподібності і показали, що θ можна розрахувати, максимізуючи функцію логарифмічної правдоподібності [205]:

$$l(\theta, x) = \sum_{i=1}^{T} \log f(x_i, \theta).$$

Дюмушель визначив функцію правдоподібності наступним чином [206]:

$$L(\theta) = \prod_{k=1}^{n} S_{\alpha,\beta}\left[(X_k - \delta)/\gamma\right]/\gamma,$$

де $\theta = (\alpha, \beta, \delta, \gamma)$ опираючись на $x = (x_1, \ldots, x_n)$ для розміру вибірки n.

12.2 Хід роботи

Для демонстрації дієвості параметрів розподілу Леві в якості індикаторів (індикаторів-передвісників) крахових подій зчитаємо, наприклад, дані одного із фондових індексів Індії з Yahoo Finance:

```
# встановлення назви індексу
symbol = "^BSESN"

# встановлення діапазону з яким будемо працювати
start = "1980-01-01"
end = "2023-11-07"

# завантаження даних з Yahoo
data = yf.download(symbol, start, end)
time_ser = data['Adj Close'].copy()

# підпис по вісі Ox
xlabel = 'time, days'

# підпис по вісі Oy
ylabel = symbol

# збереження результату в текстовий документ
np.savetxt(f'{symbol}_initial_time_series.txt', time_ser.values)
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того, з яким рядом ми працюємо

```
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 до окремої змінної
```

```
xlabel = r'$\varepsilon$'           # підпис по вісі Ox
ylabel = symbol                     # підпис по вісі Oy
```

Виводимо графік досліджуваного ряду

```
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 Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

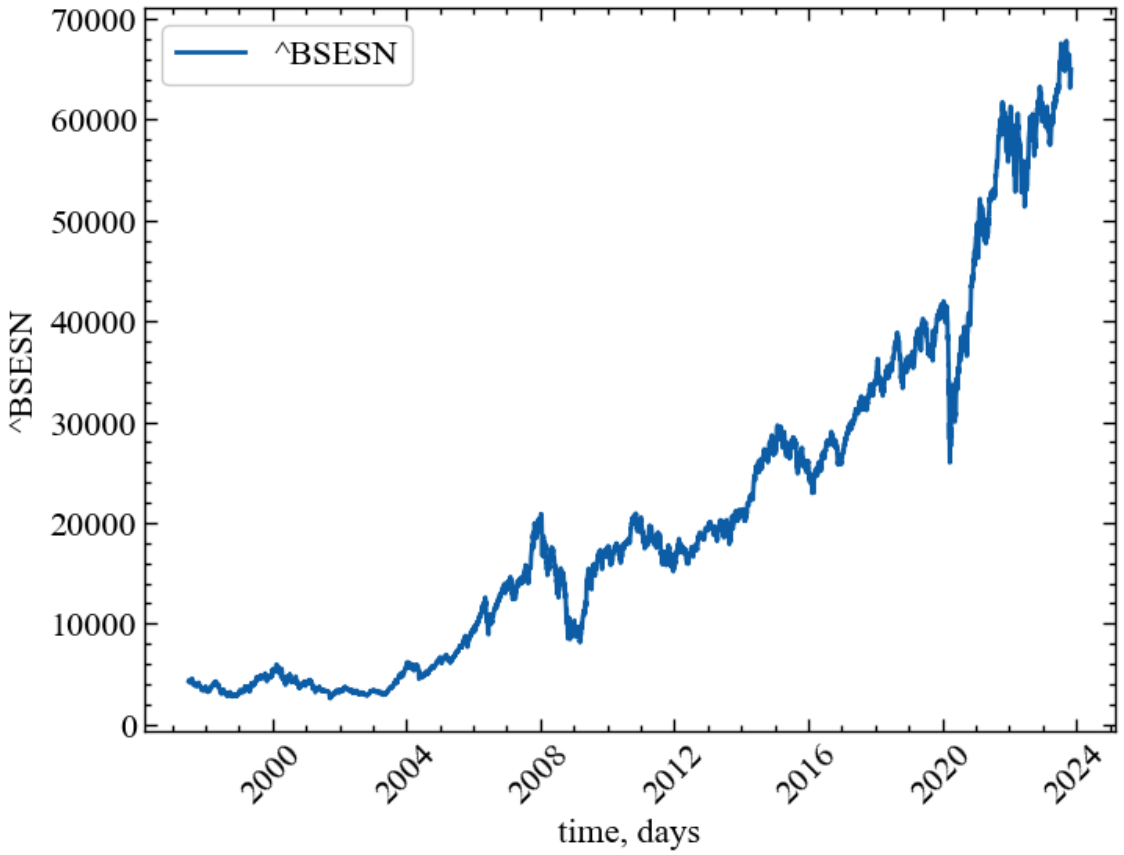


Рис. 12.17: Динаміка щоденних значень фондового індексу BSESN

12.2.1 Побудова розподілу Леві та розрахунок параметрів для всього ряду

Для приведення ряду до стандартизованого вигляду або прибутковостей визначимо функцію `transformations()`:

```
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
```

Для побудови в парі певного індикатора та досліджуваного ряду визначимо функцію `plot_pair`:

```
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', rotation=45, **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();
```

Далі виконаємо приведення ряду до прибутковостей:

```
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_levy = transformation(time_ser, ret_type)
```

Підганяємо розподіл Леві та Гауса для порівняння:

```
params = levy.fit_levy(for_levy)
mean, std = norm.fit(for_levy)
```

Отримуємо параметри розподілу Леві у відповідності до однієї із параметризацій, що пропонує пакет `levy`:

```
alpha, beta, mu, sigma = params[0].get('1')
```

Будуємо теоретичні та емпіричні розподіли:

```
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, ax = plt.subplots(1, 2)

fig.suptitle(fr'Теоретичні та емпіричні $\alpha$-стабільні розподіли для
{symbol}')

ax[0].hist(for_levy, bins=50, density=True, alpha=0.6, color='b',
label='Емпіричний')
ax[0].plot(x, pdf, 'k', label='Леві')
ax[0].plot(x, pdf_norm, 'r', label='Гаус')
ax[0].set_yscale('log')
ax[0].set_xlabel(r'$x$')
ax[0].set_ylabel(r'$f_{\alpha}(x), \, \mathrm{ePDF}$')
ax[0].legend()
```

```
ax[1].hist(for_levy, bins=50, density=True, alpha=0.6, color='g')
ax[1].plot(x, pdf, 'k')
ax[1].plot(x, pdf_norm, 'r')
ax[1].set_xlabel(r'$x$')
ax[1].set_ylabel(r'$f_{\alpha}(x), \, \mathrm{ePDF}$')

plt.savefig(f"Теоретичні та емпіричні альфа стабільні розподіли для
{symbol}.jpg")

fig.tight_layout()
plt.show();
```

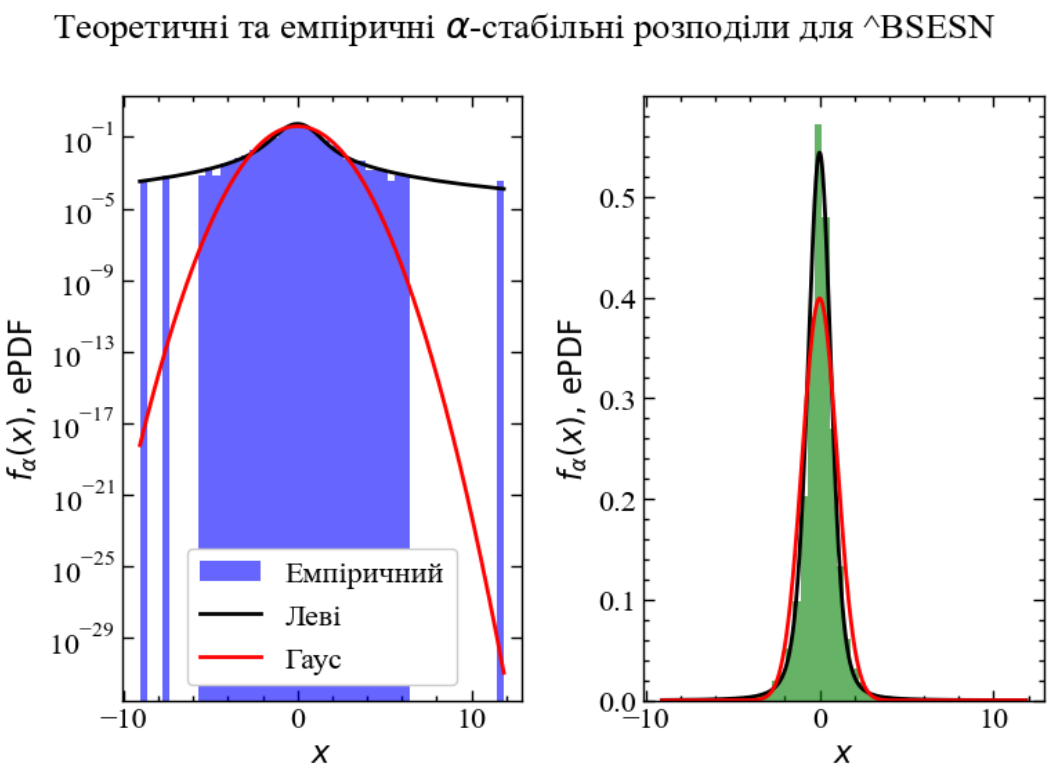


Рис. 12.1: Теоретичні та емпіричні альфа-стабільні функції щільності ймовірностей

Виводимо параметри розподілу Леві для заданого індексу

```
print(fr"Параматери alpha = {alpha:.2f}, beta = {beta:.2f}, mu = {mu:.2f}, sigma
= {sigma:.2f}")
```

```
Параматери alpha = 1.62, beta = -0.11, mu = -0.01, sigma = 0.52
```

Для досліджуваного індексу бачимо, що параметр $\alpha < 2.0$ та $\beta < 0$, що вказує на відхилення розподілу даного індексу від нормального. Тобто, для даного ряду суттєвими є кризові явища, на що вказують важкі хвости розподілу. Із порівняльного аналізу гаусового та Леві розподілів бачимо, що хвости нормального розподілу значно недооцінюють імовірність появи кризових явищ чого, наприклад, не скажеш про альфа-стабільний розподіл. Взявши логарифм

значень імовірності по осі Oy, ми можемо спостерігати, що, наприклад, недооцінка негативних прибутковостей гаусовим розподілом, у порівнянні з альфа-стабільним, складає $\approx 10^{15}$ порядків. Для позитивних прибутковостей, що перевищують значення $+10\sigma$ недооцінка гаусовим розподілом складає $\approx 10^{27}$ 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 β вказує на невелике зміщення розподілу вліво, що також демонструє переважання кризових явищ.

12.2.2 Дослідження поведінки α-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=(13, 8))

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)
ax[0][0].legend()
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 $", y=1.03)
ax[0][1].legend()
ax[0][1].set_xlabel(r"$ 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 $", y=1.03)
ax[1][0].legend(loc="lower right")
ax[1][0].set_xlabel(r"$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)
ax[1][1].legend(loc="lower right")
ax[1][1].set_xlabel(r"$x$")
ax[1][1].set_ylabel(r"$F_{\alpha}(x)$")

fig.tight_layout()
plt.show();
```

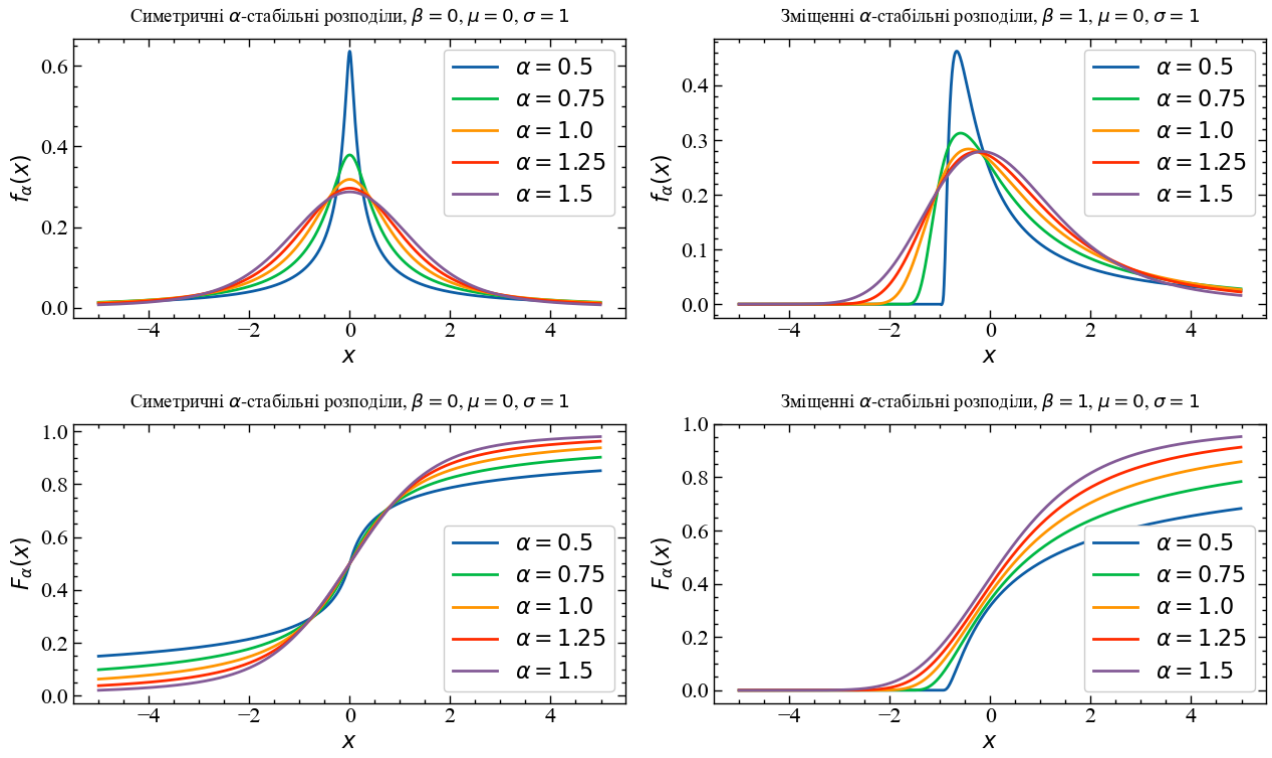


Рис. 12.2: Залежність функції щільності ймовірностей α-стабільного розподілу Леві та кумулятивної функції щільності від різних значень параметрів розподілу

12.2.3 Віконна процедура

Виконуватимемо подальші обчислення для стандартизованих прибутковостей, Покажемо, що та застосуємо α-стабільний розподіл Леві дозволяє змоделювати типові флуктуації складних систем та передчасної ідентифікації в них критичних і кризових явищ.

```
window = 250   # ширина ковзного вікна
tstep = 5      # крок

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

length = len(time_ser)

alpha = []
beta = []
mu = []
sigma = []

for i in tqdm(range(0, length-window, tstep)):
# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

# здійснюємо підгонку розподілу під значення ряду
    params = levy.fit_levy(fragm)

# отримуємо параметри розподілу
    a, b, m, s = params[0].get('0')

    alpha.append(a)
    beta.append(b)
    mu.append(m)
    sigma.append(s)
```

Зберігаємо абсолютні 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)
```

12.2.4 Динаміка показника стабільності α

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
measure_label = r'$\alpha$'
file_name = f"alpha_idx_{symbol}_{window}_{tstep}_{ret_type}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          alpha,
          ylabel,
          measure_label,
          xlabel,
          file_name)
```

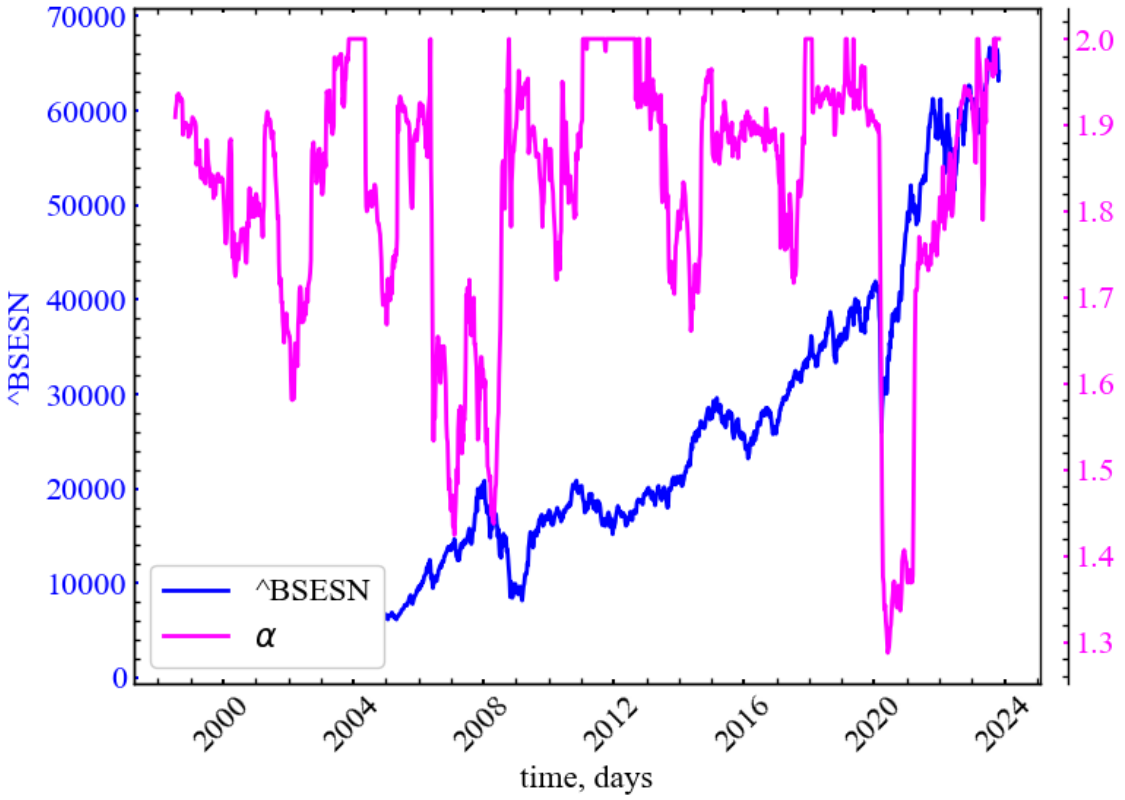


Рис. 12.3: Динаміка фондового індексу BSESN та показника стабільності α

Параметр α (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.

12.2.5 Динаміка показника асиметрії β

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
measure_label = r'$\beta$'
file_name = f"beta_idx_{symbol}_{window}_{tstep}_{ret_type}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          beta,
          ylabel,
          measure_label,
          xlabel,
          file_name)
```

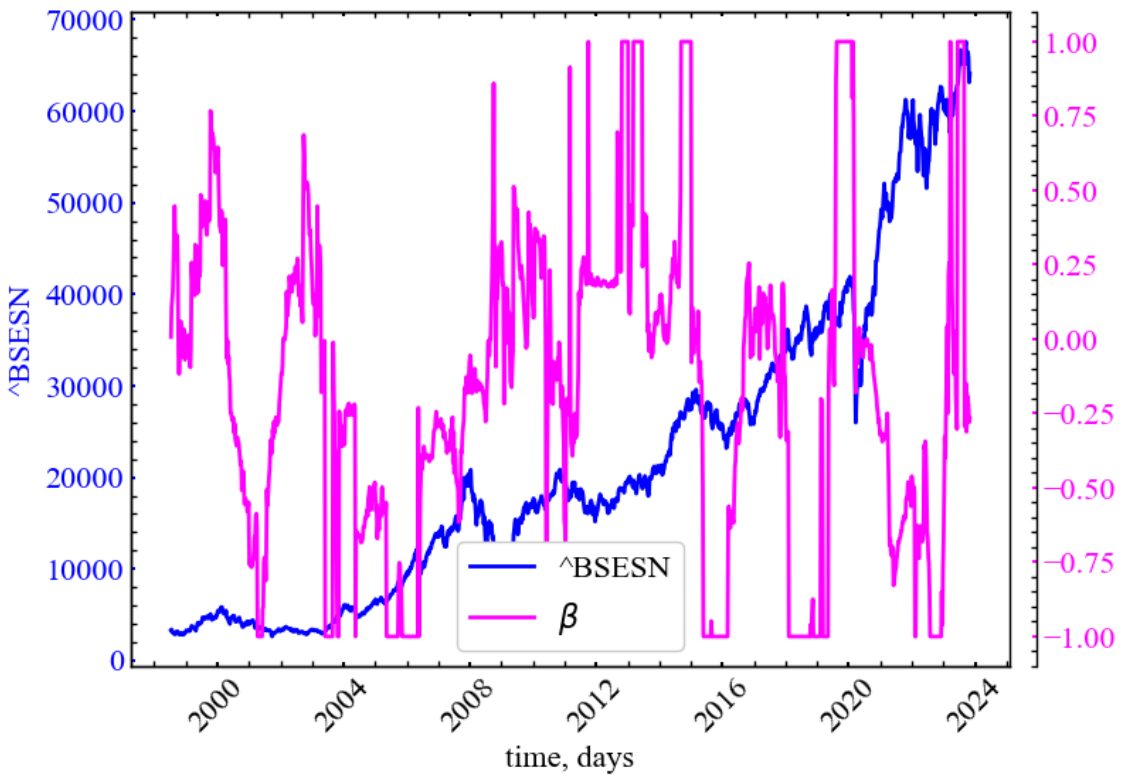


Рис. 12.4: Динаміка фондового індексу BSESN та показника асиметрії β

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.

12.2.6 Динаміка параметра зміщення μ

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
measure_label = r'$\mu$'
file_name = f"mu_idx_{symbol}_{window}_{tstep}_{ret_type}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          mu,
          ylabel,
          measure_label,
          xlabel,
          file_name)
```

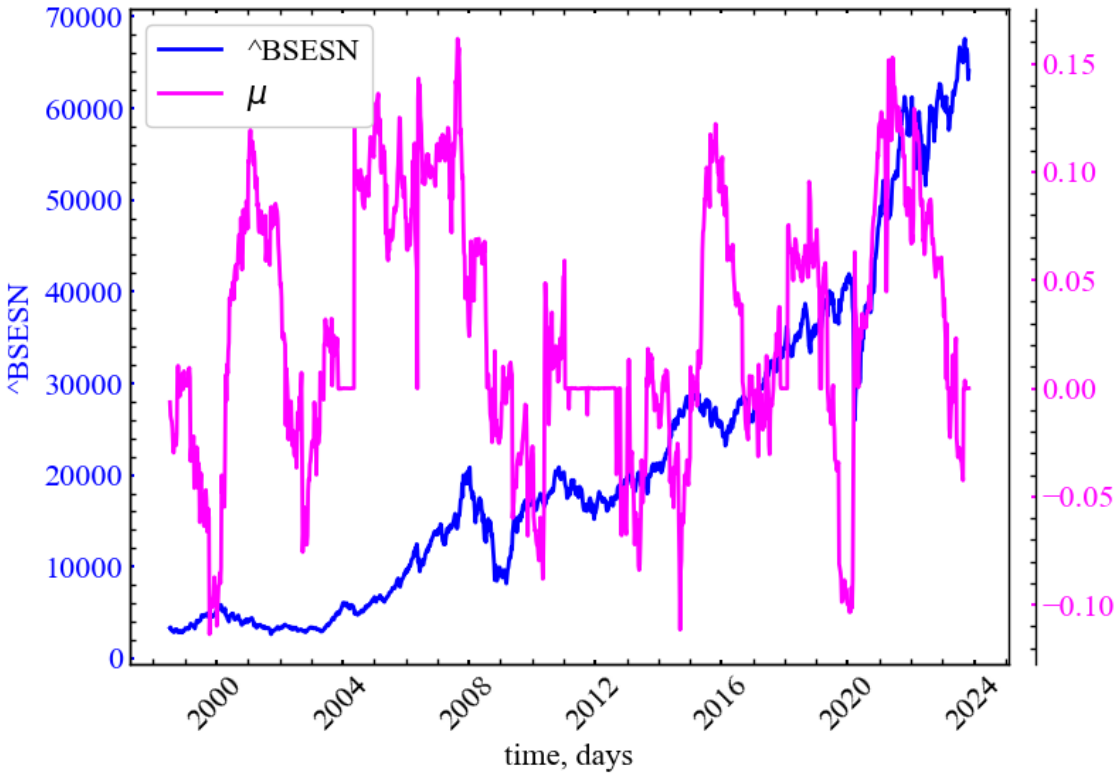


Рис. 12.5: Динаміка фондового індексу BSESN та показника зміщення μ

Показник розташування μ 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 β.

12.2.7 Динаміка параметра масштабу σ

Оголошуємо мітки для рисунків та назви збережених рисунків:

```
measure_label = r'$\sigma$'
file_name = f"sigma_idx_{symbol}_{window}_{tstep}_{ret_type}"
```

Виводимо результат:

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          sigma,
          ylabel,
          measure_label,
```

```
            xlabel,
            file_name)
```

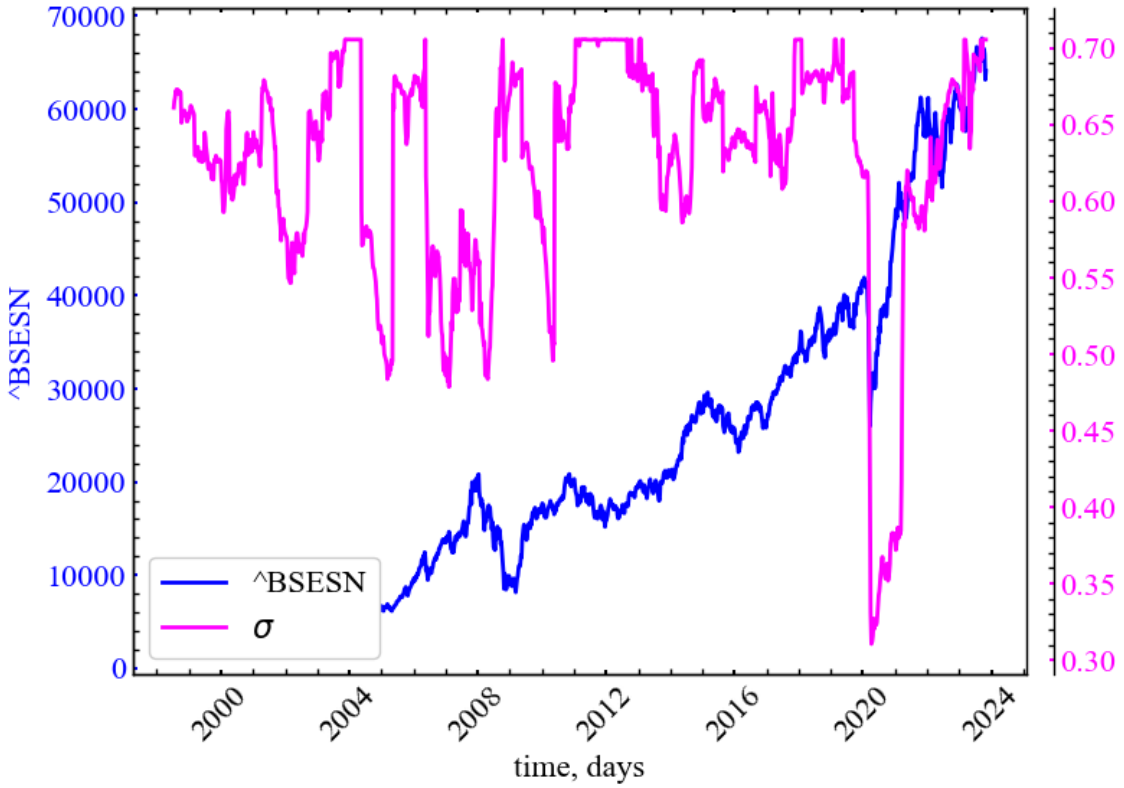


Рис. 12.6: Динаміка фондового індексу BSESN та показника масштабу σ

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 α є найкращим для ідентифікації кризових явищ та побудови надійних стратегій ризик-менеджменту.

12.3 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 α-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 α-стабільним розподілом Леві.

Використовуючи алгоритм сковзного вікна, ми побачили, що параметри α-стабільного розподілу змінюються з часом і їх можна використовувати в якості індикаторів-передвісників крахових подій.

> Література для подальшого вивчення степеневих розподілів та важких хвостів
>
> Nassim Taleb. The Black Swan: The Impact of the Highly Improbable (2007-2010 2nd. Ed.) [191]
>
> Nassim Taleb. *Statistical Consequences of Fat Tails: Real World Preasymptotics, Epistemology, and Applications* [207]

12.4 Завдання для самостійної роботи

1. Оберіть варіант ряду у викладача
2. Для вашого ряду побудуйте еміричний розподіл його прибутковостей і порівняйте його з розподілом Гауса та α-стабільним розподілом Леві. Проаналізуйте отримані результати
3. Використовуючи алгоритм ковзного вікна, дослідіть динаміку 4 ключових показників α-стабільного розподілу для вашого ряду і зробіть висновки

13. Лабораторна робота № 13

Тема. Найпростіші мережі та мережні міри складності.

Мета. Навчитися використовувати елементи теорії графів для отримання спектральний і топологічних мір складності.

13.1 Теоретичні відомості

Для сучасних складних систем характерна нерегулярність зв’язків і висока чисельність елементів, яка може досягати десятків і сотень тисяч. Таким системам та їх мережним моделям, які володіють нетривіальними топологічними властивостями, найбільше відповідає термін “комплексні”. Комплексною мережею вважається система, яка

- складається з великої кількості компонентів;
- допускає “далекосяжні” зв’язки між компонентами;
- володіє великомасштабною (у тому числі просторово-часовою) мінливістю.

Дана мережа є графом з досить великою кількістю вузлів різної природи, що характеризуються багатовимірним кортежем ознак і динамічно мінливими зв’язками; розподіл ознак вузлів і характеристик зв’язків може бути описаний ймовірнісною моделлю (багатомірним розподілом).

Основною причиною підвищення актуальності розробок у області теорії і практики комплексних мереж є результати сучасних досліджень реальних комп’ютерних, біологічних і соціальних мереж. Властивості багатьох реальних мереж істотно відрізняються від властивостей класичних випадкових графів з рівноймовірними зв’язками між вузлами, які донедавна розглядалися в якості їх базисного математичного модельного прототипу, і тому побудову їх моделей було запропоновано здійснювати з використанням зв’язних структур і степеневих розподілів.

У теорії комплексних мереж виділяють три основні напрямки:

- дослідження статистичних властивостей, які характеризують поведінку мереж;
- створення моделей мереж;
- прогнозування поведінки при зміні структурних властивостей мереж.

Комплексні мережі використовуються для моделювання об’єктів і систем, дослідження яких іншими способами (за допомогою спостереження або

активного експерименту) недоцільні або неможливі. Комп'ютерні мережі відносяться до мереж, які постійно ростуть і розвиваються. Серед факторів, що впливають на зростання мережі в першу чергу необхідно відзначити розмір або протяжність локальної мережі, яка визначається відстанню між найвіддаленішими станціями, при якій в нормальному режимі роботи вузлів чітко розпізнаються колізії, і кількість об'єднаних у мережу комп'ютерів. Для Інтернет-мереж цей розмір називається діаметром мережі і складає приблизно 1 км відстані, що дозволяє отримати високу швидкість зв'язку та максимально можливий рівень сервісу. При зростанні мережі збільшується кількість колізій, різко падає її корисна пропускна здатність і швидкодія передавання сигналу. Обмеження мережі за довжиною є передумовою вибору структури мережі, розбиття її на окремі частини (сегменти), появи додаткових серверів з новою мережею зв'язків, проблеми генеруються в контексті технологій так званої "останньої милі". Спостерігається динаміка зростання мережі, своєрідна кластеризація, сервери виступають центрами утворених кластерів, відбувається просторове позиціонування компонент мережі у вигляді чітких ієрархічних структур.

Мережа розглядається як множина сегментів, кожен з яких закінчується точкою розгалуження або кінцевої вершиною мережі. Вершинами мережі є сервери, комутатори й кінцеві користувачі, загальну кількість яких позначимо N. Локальні комп'ютерні мережі є об'єктними прототипами графових структур і тому для їх дослідження застосовують методи теорії графів.

Моделювання мереж із використанням апарата теорії графів є важливим напрямком досліджень дискретної математики. В останні роки зросла зацікавленість дослідників до складних мереж з великою кількістю вузлів, зокрема до комп'ютерних мереж, структура яких нерегулярна, складна і динамічно розвивається в часі. Для таких мереж доводиться генерувати стохастичні графи з величезною кількістю вершин.

У загальному вигляді модель комп'ютерної мережі являє собою випадковий граф, закон взаєморозміщення ребер і вершин для якого задається розподілом ймовірностей.

У даний час найпоширенішими є два основних підходи до моделювання складних мереж:

- випадкові пуассонівські графи та узагальнені випадкові графи;
- модель "тісного світу" Воттса і Строгаца [208] та її узагальнення, еволюційна модель зростання мережі Барабаші й Альберт [209,210].

Перший передбачає генерацію випадкового графа із заздалегідь відомою кількістю вершин і заданими ймовірнісними властивостями. Його ще називають графом **Ердеша-Реньї** зі сталою кількістю вершин N. Розподіл ступенів вузлів k для цього графа визначається формулою Пуассона $P(k) = \exp^{-\langle k \rangle} \langle k \rangle^k / k!$. Побудова графа здійснюється генеруванням, коли до N відокремлених вершин послідовно додаються ребра, що з'єднують випадковим чином довільні пари вершин. Початково граф складається із сукупності малих вершин, які в процесі генерування з часом розростаються до гігантського кластера зв'язаних між собою вершин, число яких є скінченною частиною загальної кількості N. При генерації постійно зростає ймовірність зв'язування вершин, яка досягає з часом деякого критичного значення. В результаті процесу, який має характер фазового переходу, граф спонтанно розростається до гігантського кластера вершин, пов'язаних між собою, що нагадує конденсацію краплі води в перенасиченій парі.

Модель **Воттса-Строгаца** є комп'ютерною моделлю тісного світу. Її побудова зводиться до наступного: розглядається одновимірний, замкнений у кільце, періодичний ланцюг, який складається із N вершин. Спочатку кожну вершину з'єднують з іншими сусідніми, які знаходяться від неї на відстані, не більшій за k, а потім кожне ребро з певною ймовірністю m перез'єднується з довільною вершиною, що призводить до трансформації регулярного ланцюга у граф тісного світу (Рис. 13.1). Оскільки в цій моделі кількість ребер є сталою, а ймовірності реалізації графів — різні, то вона зводиться до канонічного ансамблю графів і описує реально існуючі мережі, топологія яких не є ані цілком регулярною, ані цілком випадковою.

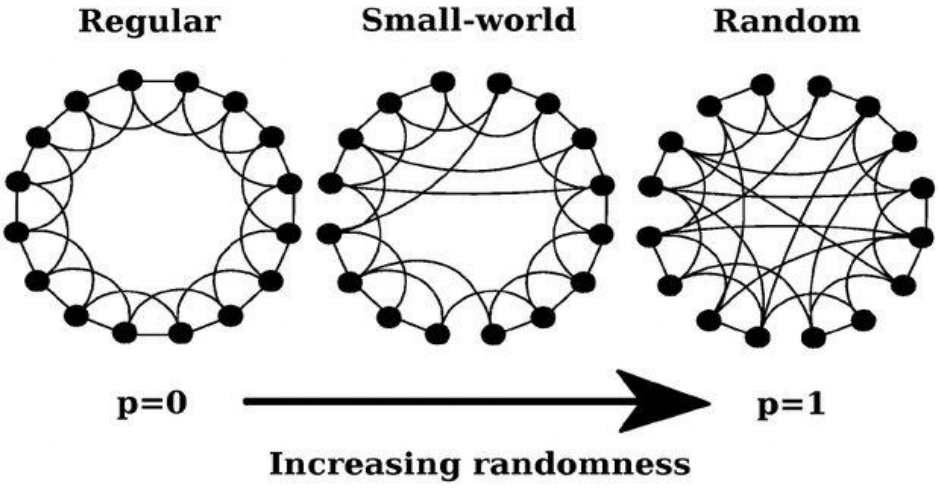


Рис. 13.1: Зростання ступеня випадковості при побудові (Increasing randomness) приводить до трансформації регулярного (Regular) ланцюга у граф тісного світу

(Small-world) і далі у випадковий (Random) граф

Більшість реальних графів підпорядковуються степеневому закону розподілу $P(k)$. Ці графи побудови мереж описуються моделлю переважного приєднання Барабаші-Альберт. Через далекосяжні взаємодії у системи не існує масштабу зміни характерних величин. Ріст і переважне приєднання є основними механізмами побудови безмасштабних (масштабно-інваріантних) мереж.

Нехай вузол i має k_i зв'язків і він може бути приєднаним (зв'язаним) до інших вузлів k_i. Ймовірність приєднання нового вузла до вузла i залежить від ступеня k_i вузла i. Величину $W(k_i) = k_i / \sum_j k_j$ називають **переважним приєднанням** (preferential attachment). Не всі вузли мають однакову кількість зв'язків, тому вони характеризуються функцією розподілу $P(k)$, що визначає ймовірність того, що випадково вибраний вузол має k зв'язків. Для складних мереж функція $P(k)$ відрізняється від розподілу Пуассона для випадкових графів. Для переважної більшості складних мереж спостерігається степенева залежність $P(k) \propto k^{-\gamma}$.

Покажемо, яким чином у межах єдиного алгоритму розрахувати і проаналізувати основні **спектральні** і **топологічні** властивості найпростіших графів. Для аналізу мережі досліджують характеристики окремих вузлів (локальні), характеристики мережі в цілому (глобальні) та характеристики мережних підструктур. Числові показники деяких глобальних характеристик мережі можуть бути представлені у вигляді аналітичних узагальнень її локальних характеристик (наприклад — найменше, найбільше, середнє значення локального показника, взяте по всім вузлам). Окрім того, що глобальна характеристика може бути представлена у формі одного числа, це також може бути представлення у вигляді розподілу значень локальної характеристики вузлів по всій мережі.

> ⓘ Примітка
>
> Частина представлених у даній лабораторній ілюстрацій була зроблена із використанням наступної книги [211]. Також у відкритому доступі наявні вихідні коди програм даної книги: https://github.com/PacktPublishing/Network-Science-with-Python-and-NetworkX-Quick-Start-Guide

13.1.1 NetworkX

Для аналізу складних мереж і їх спектральних і топологічних характеристик можна скористатися бібліотекою **NetworkX** (https://networkx.org).

NetworkX дозволяє моделювати, аналізувати та візуалізувати мережі різної природи та складності. Пакет надає класи для представлення декількох типів мереж та реалізацію багатьох алгоритмів, що використовуються в мережній науці. NetworkX відносно простий у встановленні та використанні і має багато вбудованих функцій, тому він ідеально підходить для аналізу мереж різної природи та складності.

NetworkX є безкоштовним програмним забезпеченням з відкритим вихідним кодом. Це означає, що вихідний код доступний для читання, модифікації та розповсюдження (за певних умов). Сам код доступний за адресою https://github.com/networkx/networkx.

13.1.1.1 Встановлюємо NetworkX

Для встановлення даної бібліотеки можна скористатися командою:

```
!pip install networkx
```

Далі можемо імпортувати відповідні бібліотеки:

```
import networkx as nx
import matplotlib.pyplot as plt # для візуалізації графіків
import numpy as np              # для роботи з матрицями

%matplotlib inline
```

Пам’ятайте, що оператори `import` знаходяться у верхній частині вашого коду, вказуючи Python завантажити зовнішній модуль. У цьому випадку ми хочемо завантажити NetworkX, але дамо йому короткий псевдонім `nx`, оскільки нам доведеться вводити його неодноразово, звідси й інструкція `as`.

Давайте перевіримо встановлену версію NetworkX. Ми хочемо переконатися, що не використовуємо застарілий пакет.

```
nx.__version__
```

```
'3.1'
```

Далі виконаємо налаштування формату виведення рисунків:

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
```

```
'font.size': size,                    # розмір фонтів рисунку
'lines.linewidth': 2,                 # товщина ліній
'axes.titlesize': 'small',            # розмір титулки над рисунком
'axes.labelsize': size,               # розмір підписів по осям
'legend.fontsize': size,              # розмір легенди
'xtick.labelsize': size,              # розмір розмітки по осі Ox
'ytick.labelsize': size,              # розмір розмітки по осі Ox
"font.family": "Serif",               # сімейство стилів підписів
"font.serif": ["Times New Roman"],    # стиль підпису
'savefig.dpi': 300,                   # якість збережених зображень
'axes.grid': False                    # побудова сітки на самому рисунку
}

plt.rcParams.update(params)           # оновлення стилю згідно налаштувань
```

13.1.2 Типи мереж

Почнемо з простих мереж, представлених у NetworkxX класом `Graph`.

13.1.2.1 Простий граф (ненаправлений та незважений)

```
# "звичайний" граф є неорієнтованим
G = nx.Graph()

# дайте кожній вершині "ім'я", яке у цьому випадку є літерою.
G.add_node('a')

# метод add_nodes_from дозволяє додавати вузли з послідовності, у цьому випадку
зі списку
nodes_to_add = ['b', 'c', 'd']
G.add_nodes_from(nodes_to_add)

# додаємо ребро з 'a' в 'b'
# оскільки граф неорієнтовний, то порядок не має значення
G.add_edge('a', 'b')

# так само як і add_nodes_from, ми можемо додавати ребра з послідовності
# ребра повинні бути задані як 2-кортежі
edges_to_add = [('a', 'c'), ('b', 'c'), ('c', 'd')]
G.add_edges_from(edges_to_add)

# будуємо граф

plt.figure(figsize=(6, 4))

nx.draw_networkx(G, with_labels=True)
```

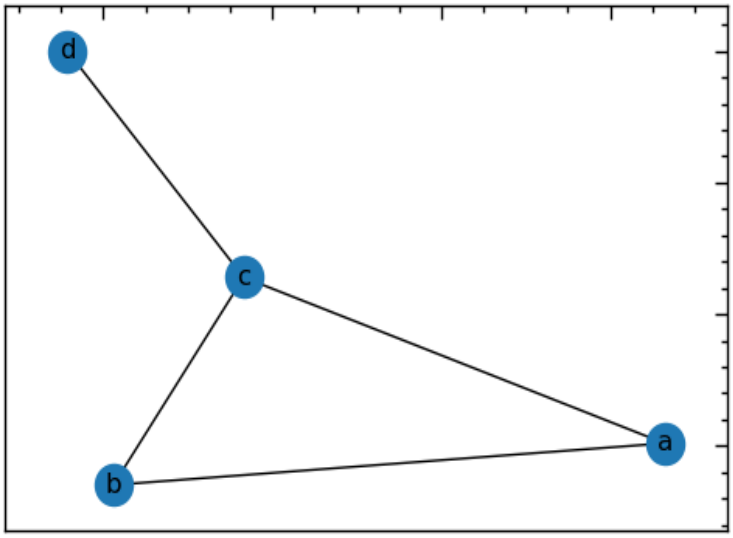


Рис. 13.2: Найпростійший граф

Існує багато необов'язкових аргументів для функції `draw_network()`, щоб налаштувати зовнішній вигляд.

```
plt.figure(figsize=(6, 4))

nx.draw_networkx(G,
        with_labels=True,
        node_color='blue',
        node_size=1600,
        font_color='white',
        font_size=16)
```

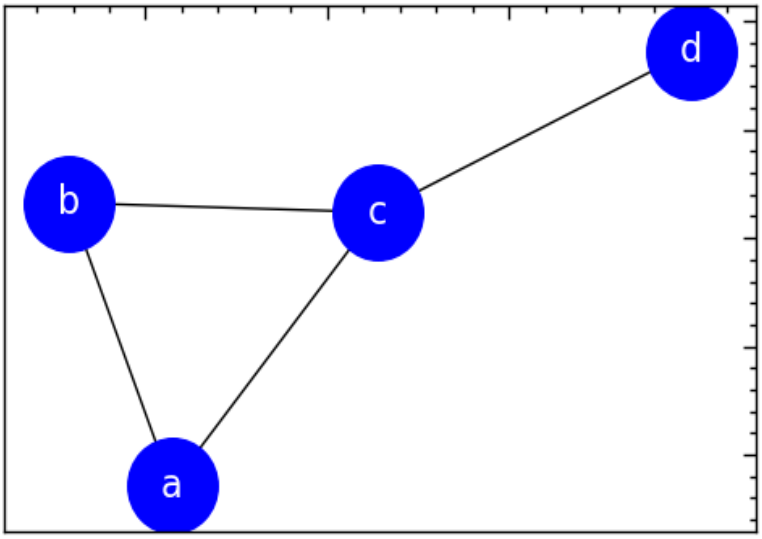


Рис. 13.3: Найпростійший граф із додатковими налаштуваннями фонтів рисунку

13.1.2.2 Зважена мережа

Повертаючись до випадку неорієнтованих мереж, іноді не всі ребра є рівними. Наприклад, у мережі, що представляє міську систему водопостачання, ребра можуть представляти серію труб, якими вода транспортується з одного місця в інше. Деякі з них можуть мати більшу пропускну здатність, ніж інші. Коли вершини графа можуть мати різну силу зв'язності, мережа називається **зваженою**, а зв'язність кількісно вимірюється числом, яке називається вагою. Зваженими можуть бути як орієнтовані, так і неорієнтовані мережі. При візуалізації мережі вагу ребер часто вказують, змінюючи товщину або непрозорість ребра. Ваги ребер можна використовувати для представлення різних типів атрибутів.

```
plt.figure(figsize=(6, 4))

# зважена мережа
G_weighted = nx.Graph()

G_weighted.add_edge("A","B",weight=6)
G_weighted.add_edge("A","D",weight=3)
G_weighted.add_edge("A","C",weight=0.5)
G_weighted.add_edge("B","D",weight=1)

nx.draw_networkx(G_weighted, with_labels=True)
```

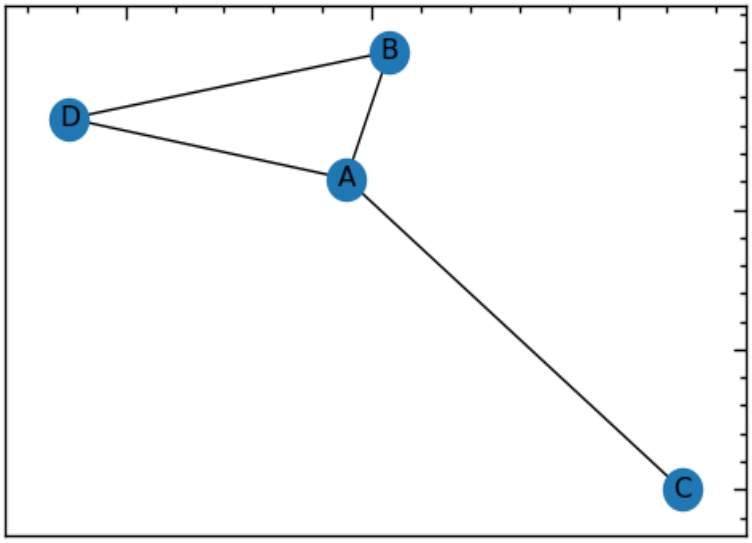


Рис. 13.4: Найпростійший зважений граф

13.1.2.3 Направлений граф

Іноді буває корисно додавати трохи більше деталей до мережі. Ребра, які ми бачили попередньо, не враховують звідки одна вершина прямує або куди. Вони

просто з’єднують два вузли, тому їх називають **симетричними** або **неорієнтованими**.

Уявіть собі мережу, яка являє собою систему доріг (ребер) і перехресть (вузлів). А мережа з ненаправленими ребрами була б гарним представленням, доки ви не натрапили на вулицю з одностороннім рухом. Ненаправлене ребро припускає, що ви можете рухатися в будь-якому напрямку однаково. Хоча в реальності напрям руху по дорожній смузі матиме значення навіть для вашого життя.

Коли напрямок має значення, мережа називається **орієнтованою (направленою)**. У направленій мережі кожне ребро має вузол-джерело і вузол-приймач. Як правило, ребро представляє якийсь потік, наприклад, трафік, від джерела до цілі. Але що, якщо не всі з’єднання є односторонніми? Двосторонні з’єднання створюються шляхом поєднання двох спрямованих ребер, що йдуть в протилежних напрямках. У спрямованих мережах ребра зображуються стрілками, що вказують на ціль.

```
plt.figure(figsize=(6, 4))

# направлений граф
G_di = nx.DiGraph()

G_di.add_edge("A","B",weight=1)
G_di.add_edge("A","D",weight=3)
G_di.add_edge("A","C",weight=1)
G_di.add_edge("B","D",weight=2)

# створити словник позицій для вузлів
pos = nx.spring_layout(G_di)

# будуємо граф
nx.draw_networkx_nodes(G_di, pos)
nx.draw_networkx_edges(G_di, pos)
nx.draw_networkx_labels(G_di, pos)

# створити словник міток ребер
edge_labels = {(u, v): d['weight'] for u, v, d in G_di.edges(data=True)}

# створення міток для ребер
nx.draw_networkx_edge_labels(G_di, pos, edge_labels=edge_labels);
```

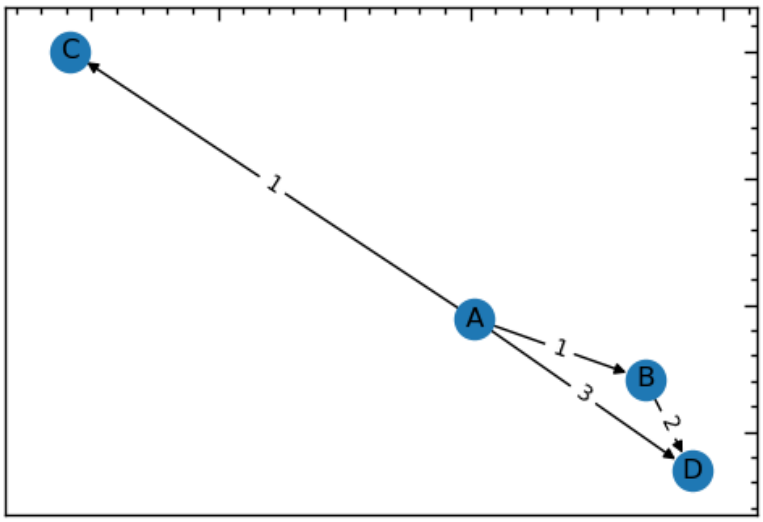


Рис. 13.5: Направлений та зважений граф

Об’єкт граф має деякі властивості та методи, які надають дані про весь граф.

```
# Список усіх вузлів
G_di.nodes()

NodeView(('A', 'B', 'D', 'C'))

# Список усіх ребер
G_di.edges()

OutEdgeView([('A', 'B'), ('A', 'D'), ('A', 'C'), ('B', 'D')])

G_di.edges(data=True) # триплет зі словником (третім йде вага ребра)

OutEdgeDataView([('A', 'B', {'weight': 1}), ('A', 'D', {'weight': 3}), ('A',
'C', {'weight': 1}), ('B', 'D', {'weight': 2})])

G_di.edges["A","B"] # виводимо вагу, вказуючи цікаві для нас вузли напряму. У
результаті отримуємо словник

{'weight': 1}

G_di.edges["A","C"]["weight"] # виводимо вагу, вказуючи цікаві для нас вузли
напряму. У результаті отримуємо скаляр

1

pos # виводимо словник координат розташувань вузлів на графіку

{'A': array([ 0.0080052 , -0.05428409]), 'B': array([ 0.3455655 , -0.29243526]),
'D': array([ 0.43837117, -0.65328066]), 'C': array([-0.79194188,  1.        ])}
```

Об’єкти NodeView та EdgeView мають ітератори, тому ми можемо використовувати їх у циклах `for`:

```
for node in G_di.nodes:
    print(node)
```

```
A
B
D
C
```

```
for edge in G_di.edges:
    print(edge)
```

```
('A', 'B')
('A', 'D')
('A', 'C')
('B', 'D')
```

Зверніть увагу, що ребра подано у вигляді 2-кортежів, так само, як ми їх ввели.

Ми можемо отримати кількість вершин та ребер у графі за допомогою методів `number_of_`.

```
G_di.number_of_nodes()
```

```
4
```

```
G_di.number_of_edges()
```

```
4
```

Деякі методи роботи з графами приймають ребро або вершину як аргумент. Вони надають властивості графа для даного ребра або вершини. Наприклад, метод `.neighbors()` повертає вершини, пов'язані з даною вершиною:

```
# список сусідів вершини 'A'
G_di.neighbors('A')
```

```
<dict_keyiterator at 0x2327b03fce0>
```

З міркувань продуктивності багато методів для роботи з графами повертають ітератори замість списків. Їх зручно використовувати у циклах:

```
for neighbor in G_di.neighbors('A'):
    print(neighbor)
```

```
B
D
C
```

І ви завжди можете використати конструктор `list` для створення списку з ітератора:

```
list(G_di.neighbors('A'))
```

```
['B', 'D', 'C']
```

Зверніть увагу на асиметрію в методах роботи з ребрами, таких як `has_edge()`:

```
G_di.has_edge('A', 'B')
```

```
True
```

```
G_di.has_edge('B', 'A')
```

```
False
```

Замість симетричного зв’язку “сусіди”, вузли в орієнтованих графах мають **попередників** (successors або “in-neighbours”) і **наступників** (predecessors або “out-neighbours”):

```
print('Попередники вершини B:', list(G_di.successors('B')))

print('Наступники вершини B:', list(G_di.predecessors('B')))
```

```
Попередники вершини B: ['D']
Наступники вершини B: ['A']
```

Спрямовані графи мають **вхідні ступені вершини** (in-degree) та **вихідні ступені вершини** (out-degree), які показують кількість ребер, що ведуть до та від даної вершини, відповідно:

```
G_di.in_degree('A') # у вершину A не входить жодна вершина (шлях)
```

```
0
```

```
G_di.out_degree('A') # з вершини A виходять 3 вершини (шляхи)
```

```
3
```

У NetworkX існує декілька алгоритмів компонування, які можуть бути використані для розміщення вузлів графа при візуалізації, в тому числі

1. nx.spring_layout(): цей алгоритм використовує примусовий підхід до розміщення вершин. Вузли, які з’єднані ребрами, притягуються один до одного, тоді як вузли, які не з’єднані, відштовхуються. Алгоритм намагається мінімізувати енергію системи, регулюючи положення вузлів.
2. nx.circular_layout(): алгоритм розміщує вузли рівномірно по колу.
3. nx.spectral_layout(): даний алгоритм використовує власні вектори матриці суміжності графа для розміщення вершин. Власні вектори використовуються для проектування вершин у простір нижчої розмірності, а положення вершин потім визначаються шляхом оптимізації функції вартості.
4. nx.random_layout(): вершини розміщуються випадковим чином у заданій обмежувальній області.
5. nx.shell_layout(): алгоритм фіксує вершини у вигляді концентричних кіл або оболонок, причому вершини в одній і тій же оболонці мають однакову відстань до центру.

6. nx.kamada_kawai_layout(): використовується ітераційний оптимізаційний підхід для розміщення вузлів. Алгоритм намагається мінімізувати навантаження на систему, змінюючи положення вузлів.
7. nx.fruchterman_reingold_layout(): варіація алгоритму nx.spring_layout(), і використання силового підходу до розміщення вузлів.

Кожен алгоритм компонування має свої сильні та слабкі сторони, і вибір найкращого з них залежить від характеристик графа та цілей візуалізації.

13.1.2.4 Знакова мережа

```
Signed_G = nx.Graph()

# додаємо ребра
Signed_G.add_edge("A","B",sign="*")
Signed_G.add_edge("A","C",sign="-")
Signed_G.add_edge("A","d",sign="+")

# вибір алгоритму компонування
pos = nx.random_layout(Signed_G)

# створити словник кольорів ребер на основі знаку кожного ребра
edge_colors = {'+':'green', '-':'red', "*":"black"}
colors = [edge_colors[Signed_G[u][v]['sign']] for u, v in Signed_G.edges()]

# створити словник стилів ребер на основі знаку кожного ребра
edge_styles = {'+':'solid', '-':'dashed', "*":"dashed"}
styles = [edge_styles[Signed_G[u][v]['sign']] for u, v in Signed_G.edges()]

plt.figure(figsize=(6, 4))

# будуємо граф з кольоровими та стилізованими ребрами
nx.draw_networkx_nodes(Signed_G, pos, node_color='blue')
nx.draw_networkx_edges(Signed_G, pos, edge_color=colors, style=styles)
nx.draw_networkx_labels(Signed_G, pos);
```

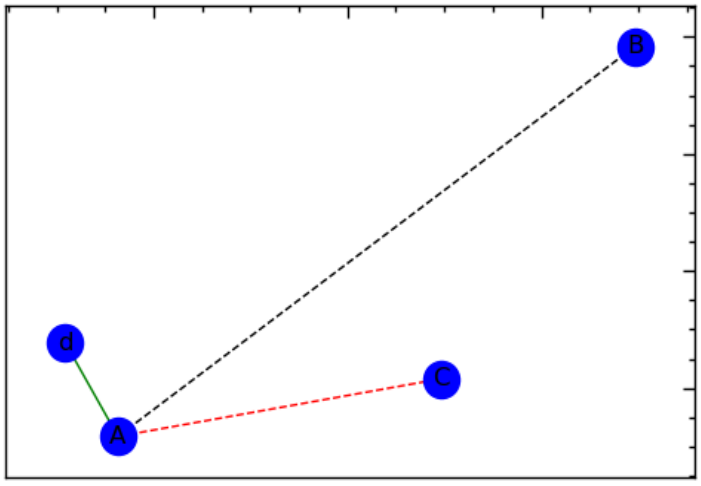


Рис. 13.6: Знакова мережа з налаштуваннями кольорів на основі знаку кожного

ребра

13.1.2.5 Мультиграф

Мультиграф — це тип графа в NetworkX, який допускає декілька ребер між парою вузлів. Іншими словами, `MultiGraph` може мати паралельні ребра, в той час як стандартний `Graph` може мати лише одне ребро між будь-якою парою вузлів. Мультиграф — це мережа, в якій декілька ребер можуть з'єднувати одні й ті ж вузли.

```
Multi_G = nx.MultiGraph()

Multi_G.add_edge("A","B",relation="family",weight=1)
Multi_G.add_edge("A","C",relation="family",weight=2)
Multi_G.add_edge("A","B",relation="Work",weight=3)
Multi_G.add_edge("D","B",relation="Work",weight=1)
Multi_G.add_edge("B","E",relation="Friend",weight=2)

plt.figure(figsize=(6, 4))

nx.draw_networkx(Multi_G, with_labels=True);
```

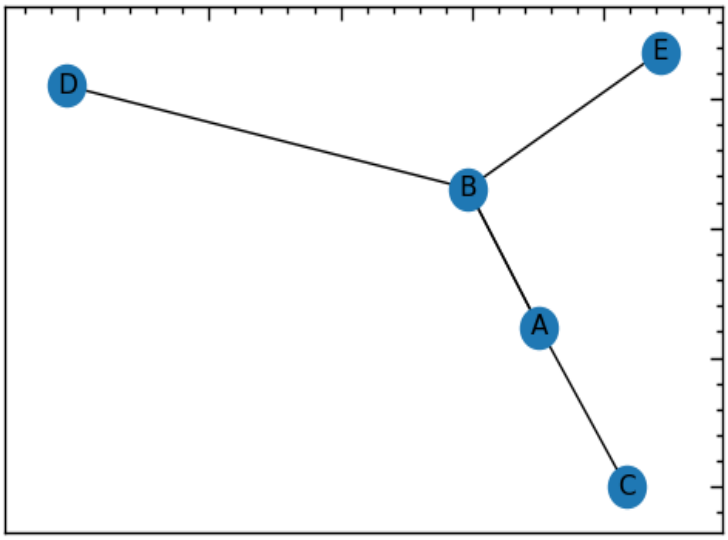


Рис. 13.7: Зважений мультиграф

```
plt.figure(figsize=(6, 4))

# компонуємо
pos = nx.spring_layout(Multi_G)

# будуємо вузли
nx.draw_networkx_nodes(Multi_G, pos, node_color='lightblue', node_size=500)

# будуємо ребра
edge_labels = {}
for u, v, d in Multi_G.edges(data=True):
```

```
    if (u, v) in edge_labels:
        edge_labels[(u, v)] += "\n" + d["relation"] + ": " + str(d["weight"])
    else:
        edge_labels[(u, v)] = d["relation"] + ": " + str(d["weight"])

nx.draw_networkx_edge_labels(Multi_G, pos, edge_labels=edge_labels)
nx.draw_networkx_edges(Multi_G, pos, width=1, alpha=0.7)

# будуємо мітки
nx.draw_networkx_labels(Multi_G, pos, font_size=10, font_family="sans-serif");
```

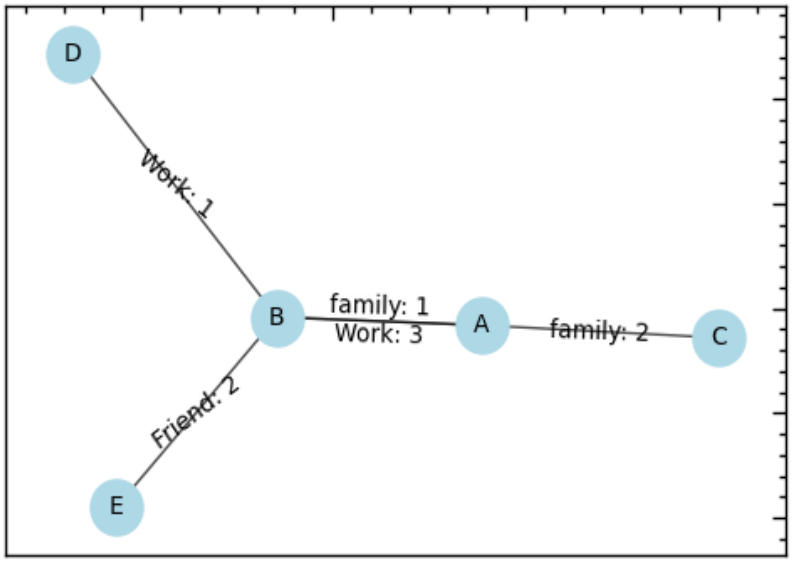


Рис. 13.8: Зважений мультиграф із покращеною візуалізацією

```
# ребра
list(Multi_G.edges())

[('A', 'B'), ('A', 'B'), ('A', 'C'), ('B', 'D'), ('B', 'E')]

# G.edges(data=True)
list(Multi_G.edges(data=True))

[('A', 'B', {'relation': 'family', 'weight': 1}),
 ('A', 'B', {'relation': 'Work', 'weight': 3}),
 ('A', 'C', {'relation': 'family', 'weight': 2}),
 ('B', 'D', {'relation': 'Work', 'weight': 1}),
 ('B', 'E', {'relation': 'Friend', 'weight': 2})]

# конкретно перелічуючи ребра
list(Multi_G.edges(data="relation"))

[('A', 'B', 'family'),
 ('A', 'B', 'Work'),
 ('A', 'C', 'family'),
 ('B', 'D', 'Work'),
 ('B', 'E', 'Friend')]

# певне ребро
Multi_G.edges["A","B"] # помилка
```

```
ValueError: not enough values to unpack (expected 3, got 2)
```

```
#натомість
# атрибути в мультиграфі

dict(Multi_G["A"]["B"])
```

```
{0: {'relation': 'family', 'weight': 1}, 1: {'relation': 'Work', 'weight': 3}}
```

```
list(Multi_G.edges("B"))
```

```
[('B', 'A'), ('B', 'A'), ('B', 'D'), ('B', 'E')]
```

```
Multi_G["A"]["B"][0]["relation"]
```

```
'family'
```

13.1.2.6 Двочастковий (bipartite) граф

Двочастковий граф — це тип графа, в якому вершини можна розбити на дві непересічні множини так, що всі ребра з'єднують вершину з однієї множини з вершиною в іншій множині. Тобто, не існує ребер, які з'єднують вершини всередині однієї множини.

Двочасткові графи корисні для моделювання відносин між двома різними типами об'єктів, наприклад, покупцями і продавцями на ринку, або акторами і фільмами в кіноіндустрії.

У NetworkX ви можете створювати і маніпулювати двосторонніми графами за допомогою модуля `bipartite`, який надає різні функції і алгоритми для двосторонніх графів. Крім того, існує декілька методів візуалізації, які можна використовувати для відображення двосторонніх графів, наприклад, двосторонній макет, який розташовує вузли у два окремі рядки.

Приклад акціонерів та акцій:

```
from networkx.algorithms import bipartite

# список акціонерів
stockholders = ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace',
'Harry', 'Ivy', 'John']

# перелік акцій
stocks = ['AAPL', 'GOOG', 'TSLA', 'AMZN', 'FB', 'MSFT', 'NVDA', 'PYPL', 'NFLX',
'TWTR']

# створити двочастковий граф
B = nx.Graph()

# додавання вузлів зі списку
B.add_nodes_from(stockholders, bipartite=0)
B.add_nodes_from(stocks, bipartite=1)

# додавання ребер випадковим чином
```

```
import random
while not nx.is_connected(B):
    B.add_edge(random.choice(stockholders), random.choice(stocks))

plt.figure(figsize=(8, 6))

# будуємо двочастковий граф
pos = nx.bipartite_layout(B, stockholders)
nx.draw_networkx(B, pos, with_labels=True)
```

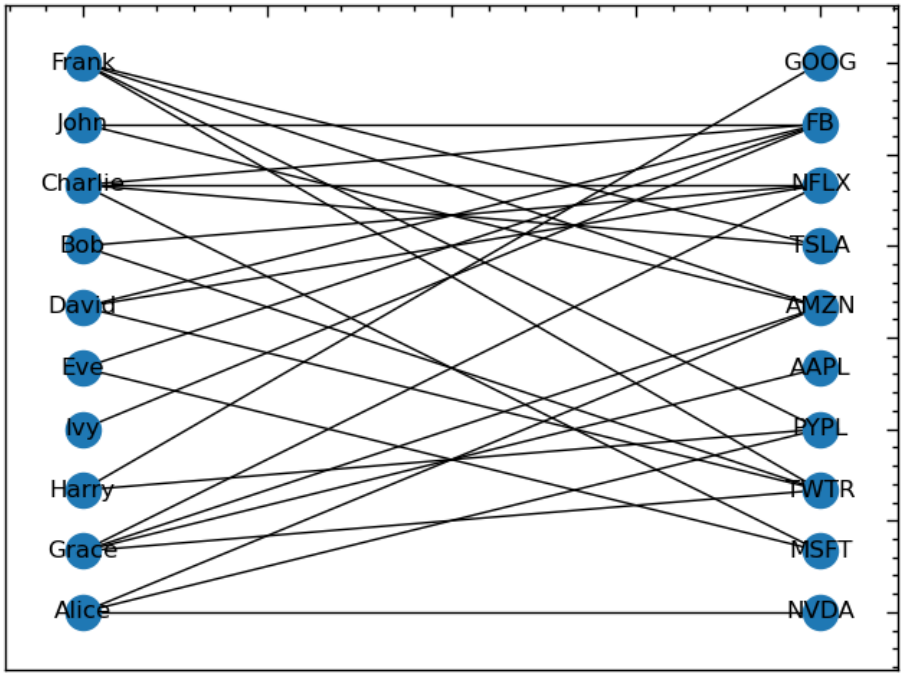


Рис. 13.9: Двочасткова мережа акціонерів та їх акцій

```
# 2 набори двочасткових графів
bipartite.sets(B)
```

```
({'Alice','Bob','Charlie', 'David', 'Eve', 'Frank', 'Grace', 'Harry', 'Ivy',
'John'}, {'AAPL', 'AMZN', 'FB', 'GOOG', 'MSFT', 'NFLX', 'NVDA', 'PYPL', 'TSLA',
'TWTR'})
```

Хоча двочасткові графи корисні для представлення повної структури зв'язків "багато-до-багатьох", іноді простіше працювати зі стандартними односторонніми мережами. Це може бути у випадку, якщо аналіз фокусується на певному типі вузлів, або якщо необхідна методика доступна лише для односторонніх (одномодальних) мереж, або ж методика доступна лише для одномодових мереж, чи мережа зв'язків має занадто багато вузлів для чіткої візуалізації. На щастя, можна створити одномодові мережі з мережі зв'язків за допомогою процесу, який називається "проєкція".

Одномодові мережі, побудовані з мереж зв'язків, називаються мережами спільної приналежності, тому що вузли з'єднуються ребрами, якщо вони мають

спільні зв’язки. Існує кілька типів проекцій, які використовуються для створення спільної приналежності, але всі вони обертаються навколо однієї і тієї ж ідеї: з’єднання вузлів зі спільним сусідом у вихідній мережі приналежності. Найпростіша можлива проекція — це незважена проекція, яка створює незважене ребро між вузлами з одним або декількома спільними сусідами. Наступний код використовує функцію projected_graph() для проектування мережі акціонерів, що мають спільні акції компаній:

```
# Лівобічний граф (акціонери)
# акціонери, які мають спільні акції, пов'язані між собою
# Лівобічний граф
plt.figure(figsize=(6, 4))

P = bipartite.projected_graph(B, bipartite.sets(B)[0])
nx.draw_networkx(P, with_labels=True, node_size=10)
```

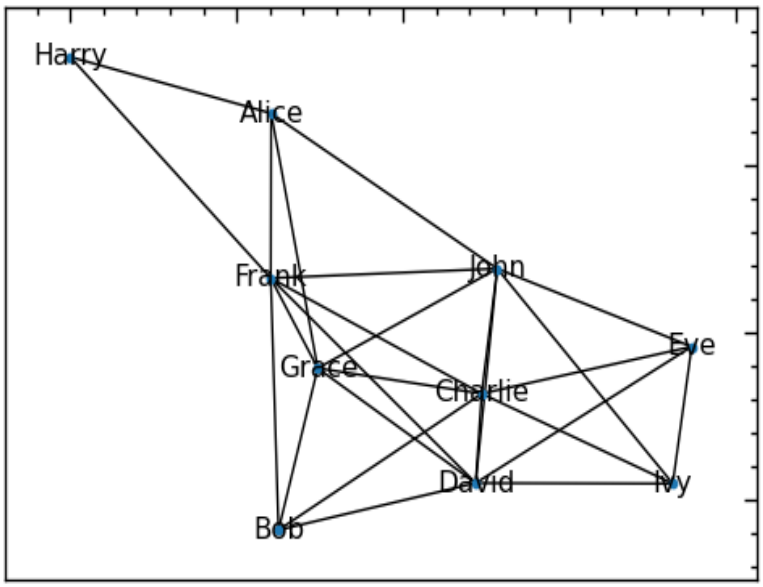


Рис. 13.10: Лівобічний граф мережі акціонерів у котрих наявні спільні акції

У такий самий спосіб ми можемо побудувати мережу акцій:

```
# Правобічний граф
plt.figure(figsize=(6, 4))

P = bipartite.projected_graph(B,bipartite.sets(B)[1])
nx.draw_networkx(P, with_labels=True, node_size=10)
```

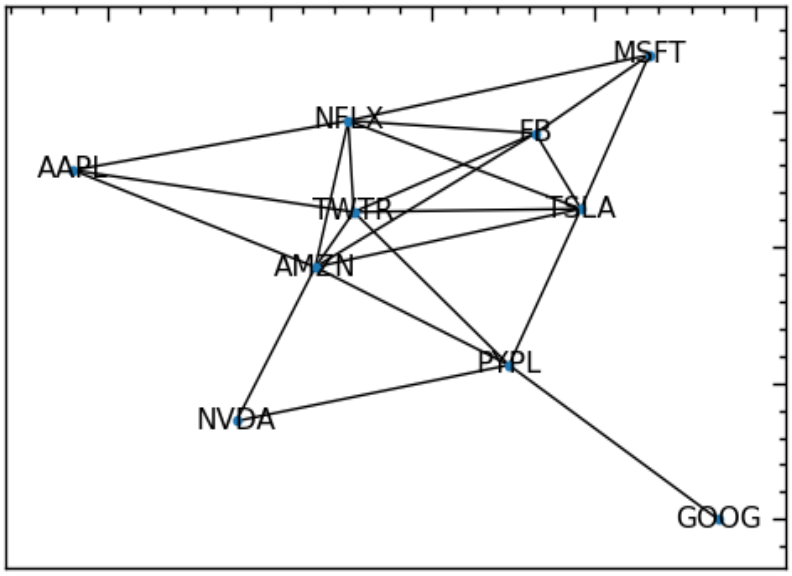


Рис. 13.11: Лівобічний граф акцій у котрих наявні спільні акціонери

```
# Зважений лівобічний граф
# як багато спільного

plt.figure(figsize=(6, 4))

# краща візуалізація
P = bipartite.weighted_projected_graph(B,bipartite.sets(B)[0])
pos = nx.circular_layout(P)

# будуємо граф
nx.draw_networkx_nodes(P, pos)
nx.draw_networkx_edges(P, pos)
nx.draw_networkx_labels(P, pos)

# створюємо словник міток ребер
edge_labels = {(u, v): d['weight'] for u, v, d in P.edges(data=True)}

# будуємо мітки для ребер
nx.draw_networkx_edge_labels(P, pos, edge_labels=edge_labels);
```

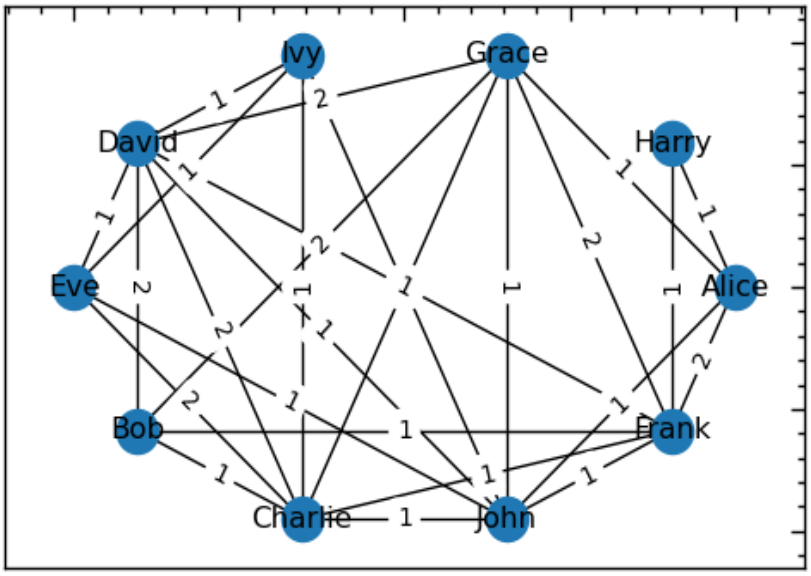


Рис. 13.12: Лівобічний граф мережі акціонерів, де вагові коефіцієнти на ребрах указують на кількість наявних спільних акцій між акціонерами

13.1.3 Імпортуємо інформацію про мережу

13.1.3.1 Імпортуємо дані з file.txt та GEXF

Щоб імпортувати інформацію про мережу до NetworkX, ви можете скористатися однією з декількох функцій, залежно від формату ваших даних. Ось кілька прикладів:

1. Імпорт з файлу списку граней:

Припустимо, у вас є файл списку граней, що мають наступне представлення:

```
A B
A C
B D
C D
D E
```

Ви можете імпортувати цей файл у граф NetworkX функцією `read_edgelist()`:

```
G = nx.read_edgelist('databases\lab_13\Sample1.txt')

plt.figure(figsize=(6, 4))
nx.draw_networkx(G, with_labels=True)
```

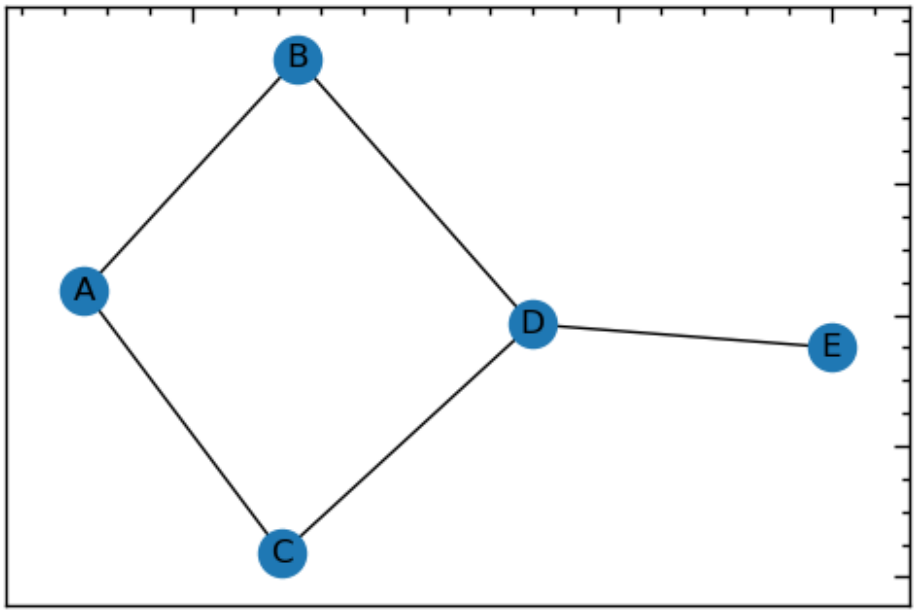


Рис. 13.13: Простий граф, що був зчитаний з текстового файлу за допомогою функції `read_edgelist()`

2. Імпорт з файлу матриці суміжності:

Припустимо, що у вас є файл матриці суміжності:

```
0 1 1 0 0
1 0 0 1 0
1 0 0 1 1
0 1 1 0 1
0 0 1 1 0
```

Ви можете імпортувати цей файл у граф NetworkX за допомогою функції `from_numpy_array()`:

```
adj_matrix = np.loadtxt('databases\lab_13\Sample2.txt')

G = nx.from_numpy_array(adj_matrix)

plt.figure(figsize=(6, 4))
nx.draw_networkx(G, with_labels=True)
```

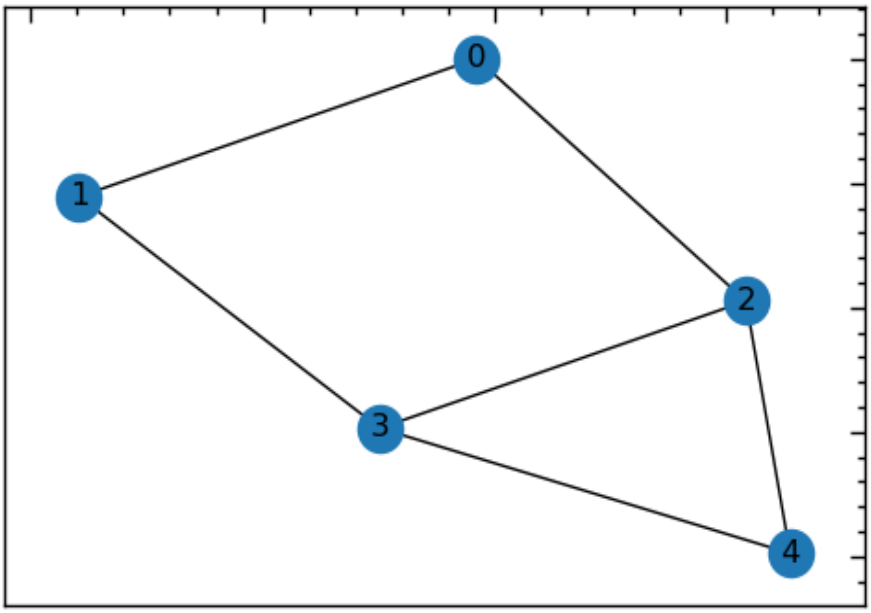


Рис. 13.14: Простий граф, що був зчитаний з масиву `numpy` за допомогою функції `from_numpy_array()`

3. Імпорт з файлу GEXF:

Якщо у вас є файл мережі у форматі GEXF, який є популярним форматом для обміну даних про графи між різними програмними пакетами, ви можете імпортувати його у граф NetworkX за допомогою функції `read_gexf`:

Простий граф у форматі GEXF:

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.3" version="1.3">
<meta lastmodifieddate="2022-10-01">
<creator>NetworkX</creator>
<description>An example graph in GEXF format</description>
</meta>
<graph mode="static" defaultedgetype="undirected">
<nodes>
<node id="0" label="Node 0"/>
<node id="1" label="Node 1"/>
<node id="2" label="Node 2"/>
</nodes>
<edges>
<edge id="0" source="0" target="1"/>
<edge id="1" source="1" target="2"/>
<edge id="2" source="2" target="0"/>
</edges>
</graph>
</gexf>

G = nx.read_gexf('databases\lab_13\\basic.gexf')

plt.figure(figsize=(6, 4))
nx.draw_networkx(G, with_labels=True)
```

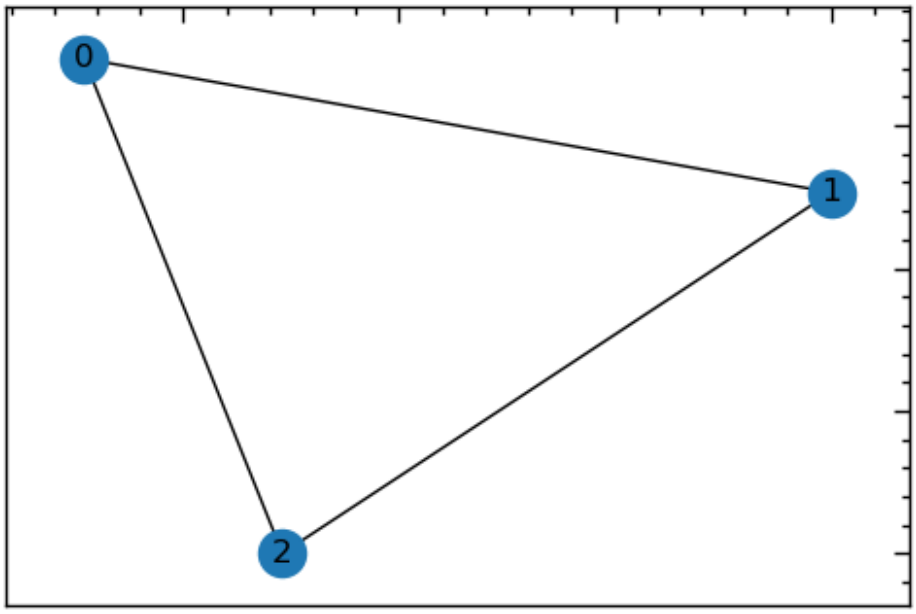


Рис. 13.15: Простий граф, що був зчитаний з файлу формату `.gexf` за допомогою функції `read_gexf()`

```
# Зберігаємо граф у форматі GEXF
nx.write_gexf(G, 'databases\lab_13\Sample3.gexf')
```

13.1.3.2 Матриця суміжності

```
G_mat = np.array([[0, 1, 1, 1, 0, 1, 0, 0, 0, 0],
                  [1, 0, 0, 1, 0, 0, 1, 0, 0, 0],
                  [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [1, 1, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 1, 0, 1, 0, 1, 0, 0],
                  [1, 0, 0, 0, 1, 0, 0, 0, 1, 0],
                  [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
                  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]])

G_mat
```

```
array([[0, 1, 1, 1, 0, 1, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 1, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 1, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 1, 0, 1, 0, 0],
       [1, 0, 0, 0, 1, 0, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]])
```

Перетворення матриці суміжності у граф за допомогою `nx.Graph`:

```
G = nx.Graph(G_mat)
```

```
plt.figure(figsize=(6, 4))
nx.draw_networkx(G)
```

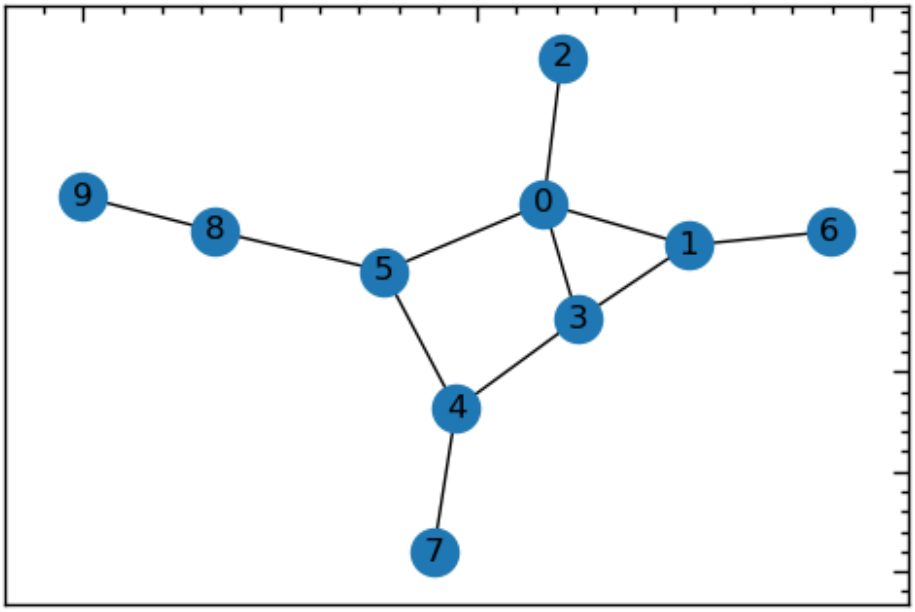


Рис. 13.16: Простий граф, що був побудований напряму з матриці суміжності

13.1.4 Графостатистичні показники

13.1.4.1 Ступінь вершини

Незалежно від того, чи представляють вузли людей, місця, комп'ютери або атоми, розташування вузла в структурі мережі тісно пов'язане з роллю, яку він відіграє в загальній системі. Різні структури уможливлюють різні ролі. Отже, кількісно оцінюючи структурні властивості вузла, можна зрозуміти роль, яку відіграє цей вузол. Числові міри, які характеризують мережеві властивості вузла, називаються мірами центральності. Однією з найпростіших мір центральності є **ступенева центральність** (degree centrality). Ступенева центральність вузла — це просто кількість сусідів, які наявні у вузла. У соціальній мережі ступенева центральність є мірою популярності, і може бути хорошим способом здогадатися, хто влаштовує найкращі вечірки, хто має найбільшу кількість публікацій або хто є монополістом на ринку праці. Ступенева центральність — це досить елементарний приклад, але далі будуть представлені більш складні міри, які часто використовуються в науці складних мереж. Кожна міра центральності кількісно оцінює різний тип важливості і може бути корисною для відповідей на різні типи питань.

Показник ступеневої центральності тісно пов'язаний із такою мірою як **ступінь вершини в мережі** (node degree), яка визначає кількість ребер, з якими

з’єднана конкретна досліджувана вершина. У мережі з N вершин і M ребер, ступінь k_i вершини i визначається як

$$k_i = \sum_{j=1}^{M} A_{ij}.$$

A — матриця суміжності мережі, $A_{ij} = 1$, якщо існує ребро, що з’єднує вершини i та j, і $A_{ij} = 0$ в іншому випадку.

Околицею вершини i називається множина вершин, які безпосередньо з’єднані з i-им ребром. Околиця i позначається як N_i і визначається як

$$N_i = \{j \mid A_{ij} = 1\}.$$

Тут A — матриця суміжності мережі; $A_{ij} = 1$, якщо існує ребро, що з’єднує вершини i та j, і $A_{ij} = 0$ в іншому випадку.

Розглянемо приклад ступеню вершини на прикладі графа **карате-клубу**.

> Відомості про граф карате-клубу
>
> Граф карате-клубу — це соціальна мережа, що представляє дружбу між 34 членами карате-клубу, як це спостерігав Вейн В. Захарі у 1977 році. Кожна вершина графа представляє члена клубу, а кожне ребро — дружбу між двома членами. Граф має 34 вершини та 78 ребер. Карате-клуб є відомим прикладом аналізу соціальних мереж і використовувався для вивчення різних властивостей мережі, таких як структура спільноти і міра центральності. Граф характеризується розколом клубу на дві фракції, очолювані інструкторами клубів: вершина 1 та вершина 34. Цей розкол був спричинений суперечкою між двома лідерами, яка врешті-решт призвела до утворення двох окремих клубів карате

```
G_karate = nx.karate_club_graph()

plt.figure(figsize=(6, 4))
nx.draw_networkx(G_karate)
```

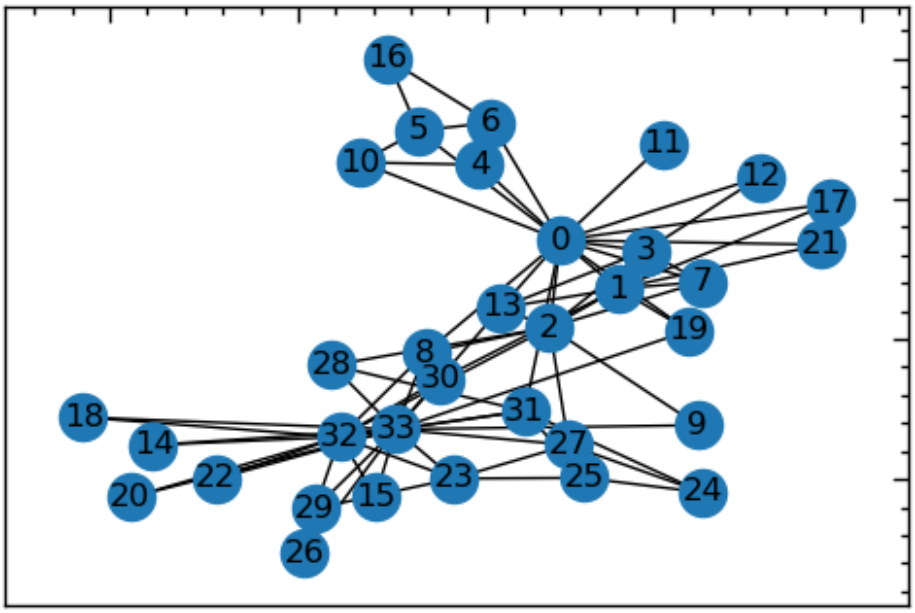


Рис. 13.17: Граф карате-клубу

```
node = 2
neighborhood = list(nx.neighbors(G_karate, node))
neighborhood
```

```
[0, 1, 3, 7, 8, 9, 13, 27, 28, 32]
```

```
# ступінь = кількість сусідів
len(neighborhood)
```

```
10
```

```
# ступінь вершини
G_karate.degree(node)
```

```
10
```

```
# Усі ступені вершини
dict(G_karate.degree)
```

```
{0: 16, 1: 9, 2: 10, 3: 6, 4: 3, 5: 4, 6: 4, 7: 4, 8: 5, 9: 2, 10: 3, 11: 1, 12:
2, 13: 5, 14: 2, 15: 2, 16: 2, 17: 2, 18: 2, 19: 3, 20: 2, 21: 2, 22: 2, 23: 5,
24: 3, 25: 3, 26: 2, 27: 4, 28: 3, 29: 4, 30: 4, 31: 6, 32: 12, 33: 17}
```

```
plt.figure(figsize=(6, 4))
plt.hist(sorted(dict(G_karate.degree).values()))
plt.xlabel("Ступінь вершини")
plt.ylabel("Частота")
```

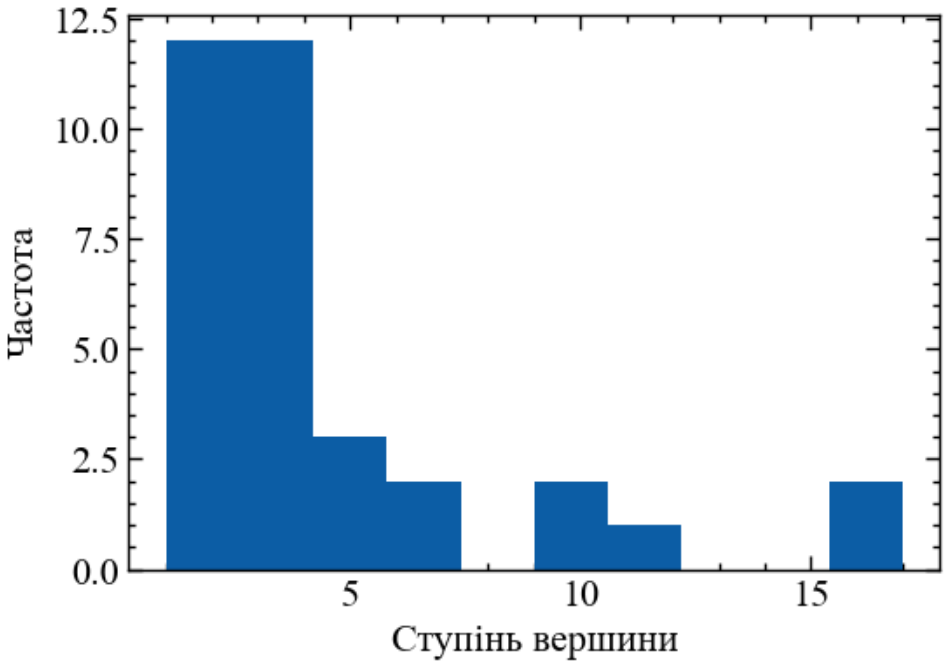


Рис. 13.18: Гістограма ступенів вершини в графі карате-клубу

На Рис. 13.18 видно, що в мережі карате-клубів наявно достатньо багато учасників клубу, хто має один або декілька зв'язків із іншими членами клубу. Також видно, що серед них є ті, хто має більше 15 знайомих. Представниками з такою кількістю зв'язків можуть бути лідери цих клубів.

13.1.4.2 Тріадичне закриття

Міра, представлена в цьому розділі, стосується зв'язків між сусідами вузла, а не самого вузла. Часто буває корисно розглянути, чи мають сусіди вузла тенденцію бути пов'язаними один з одним. У соціальній мережі це питання зводиться до того, щоб запитати, чи товариш вашого товариша є і вашим товаришем одночасно. Ця властивість відома як **транзитивність** (transitivity). Результатом таких стосунків є трикутники: три вузли, пов'язані між собою. Тенденція до виникнення таких трикутників називається **кластеризацією** (clustering). Сильна кластеризація часто свідчить про надійність і надлишковість мережі — якщо один ребро зникає, шлях все ще існує через два інших. Кластеризація вимірюється за допомогою **коефіцієнта локальної кластеризації** (local clustering coefficient), який визначає тенденцію вузлів об'єднуватись у тріади. **Глобальний коефіцієнт кластеризації** (global clustering coefficient) представляє середнє значення по всім локальним кластеризаціям, що були визначені для кожного вузла мережі.

13.1.4.2.1 Коефіцієнт кластеризації

Коефіцієнт кластеризації вершини i задається формулою:

$$C_i = \sum_{j,k} A_{ij} A_{jk} A_{ki} / k_i (k_i - 1),$$

де $k_i = \sum_j A_{ij}$ — кількість ребер, що входять у вершину i; A позначає матрицю суміжності.

```
# локальна кластеризація
nx.clustering(G, 2)
```

```
0
```

```
# список кластеризацій
nx.clustering(G)
```

```
{0: 0.16666666666666666, 1: 0.3333333333333333, 2: 0, 3: 0.3333333333333333, 4:
0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0}
```

13.1.4.2.2 Глобальний коефіцієнт кластеризації

Багато спостережуваних соціальних мереж є більш кластеризованими, ніж це могло б виникнути випадковим чином

Коефіцієнт кластеризації мережі є середнім значенням коефіцієнтів кластеризації всіх N вузлів:

$$C = N^{-1} \sum_{i=1}^{N} C_i.$$

```
# середній ступінь кластеризації
nx.average_clustering(G)
```

```
0.08333333333333333
```

13.1.4.2.3 Транзитивність

Транзитивність — це властивість мережі, яка вимірює ймовірність того, що якщо два вузли мережі мають спільного сусіда, то вони також будуть безпосередньо з’єднані один з одним. Іншими словами, вона вимірює тенденцію до утворення “трикутників” у мережі.

Формально транзитивність мережі визначається як відношення кількості трикутників у мережі до кількості з’єднаних трійок вузлів (тобто трійок вузлів, які безпосередньо з’єднані один з одним або мають спільного сусіда). У математичній нотації транзитивність мережі позначається як

$$T = \sum_{i,k,j=1}^{N} A_{ik}\, A_{kj} A_{ji} \Big/ \sum_{i,k,j=1}^{N} A_{ik}\, A_{ji}.$$

Висока транзитивність вказує на те, що вузли в мережі мають тенденцію до утворення трикутних кластерів або спільнот, тоді як низька транзитивність вказує на те, що мережа є більш випадковою або децентралізованою структурою. Транзитивність тісно пов'язана з поняттям коефіцієнта кластеризації, який вимірює схильність вузлів до утворення локальних кластерів або спільнот.

```
#транзитивність
#transitivity зважує вершини з великим ступенем вершини
nx.transitivity(G)
```

```
0.15789473684210525
```

13.1.4.3 Шлях

Шлях (path) між двома вузлами A та B у мережі — це послідовність вузлів $A, X_1, X_2, \ldots, X_n, B$ та послідовність ребер $(A, X_1), (X_1, X_2), \ldots, (X_n, B)$, де кожен вузол та ребро у послідовності є суміжним з попереднім та наступним вузлом або ребром у послідовності.

Довжина шляху (path length) — це кількість ребер у ньому. Шлях довжиною 1 — це ребро між двома вершинами, шлях довжиною 2 — послідовність з двох ребер і трьох вершин, і так далі. Найкоротший шлях між двома вершинами — це шлях мінімальної довжини, який їх з'єднує.

```
# згенерувати усі прості шляхи між вершинами 1 та 3
paths = nx.all_simple_paths(G, source=1, target=3)

# перетворити генератор у список
Path_List = [path for path in paths]

print("Список шляхів:", Path_List)
```

```
Список шляхів: [[1, 0, 3], [1, 0, 5, 4, 3], [1, 3]]
```

```
Path1 = Path_List[0]
# перевірити, чи є шлях простим у графі
is_valid = nx.is_simple_path(G, Path1) # Простий шлях - це шлях, який не містить жодної вершини, що повторюється.
print("Чи є шлях простим?", is_valid)
```

```
Чи є шлях простим? True
```

```
# хибний приклад
nx.is_simple_path(G, [0,8,5])
```

```
False
```

```
# формуємо список ребер, що формують шлях
edge_list = [(Path1[i], Path1[i+1]) for i inrange(len(Path1)-1)] # len(Path1)-1
= довжина шляху
edge_list
```

```
[(1, 0), (0, 3)]
```

```
# обчислюємо вагу шляху
weight = sum(G[u][v]['weight'] for u, v in edge_list if 'weight' in G[u][v])
print("Вага шляху:", weight)
```

```
Вага шляху: 2
```

13.1.4.3.1 Геодезична лінія

Геодезичний шлях між двома вузлами A і B в мережі — це **найкоротший шлях**, який їх з'єднує. Іншими словами, це шлях з мінімальною кількістю ребер, які потрібно пройти, щоб дістатися з вузла A до вузла B. Довжина геодезичного шляху — це кількість ребер у цьому шляху.

```
# геодезичний шлях = найкоротший шлях
nx.shortest_path(G, 1, 2)
```

```
[1, 0, 2]
```

```
# обчислити найкоротший шлях між двома вузлами
path = nx.shortest_path(G, source=1, target=3)

# обчислити відповідні ребра шляху
edges = [(path[i], path[i+1]) for i in range(len(path)-1)]

plt.figure(figsize=(6, 4))

# будуємо граф та шлях
pos = nx.circular_layout(G)
nx.draw_networkx(G, pos, with_labels=True)
nx.draw_networkx_edges(G, pos, edgelist=edges, edge_color='r', width=3);
```

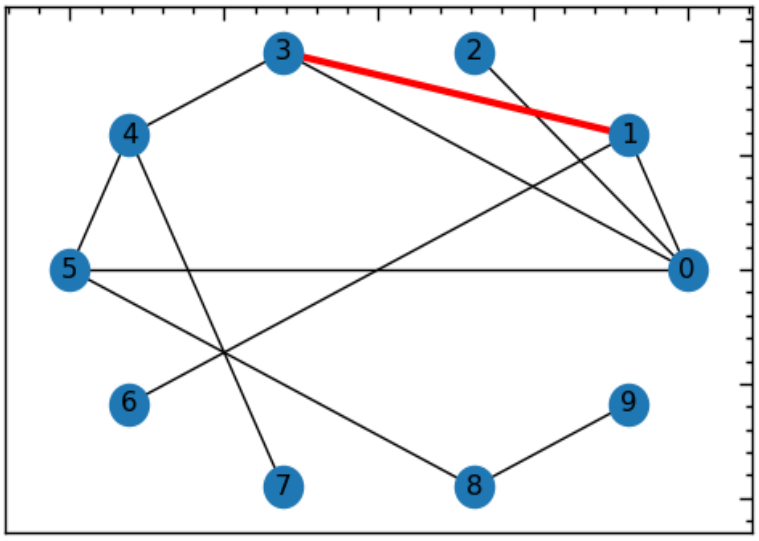


Рис. 13.19: Граф із виділеним найкоротшим шляхом між вузлами 1 і 3

```
# геодезична довжина
nx.shortest_path_length(G, 1, 2)
```

```
2
```

Пошук геодезичного шляху від вузла i до кожного іншого вузла є обчислювально складним, тому нам потрібен ефективний алгоритм для цього.

Тут ми використовуємо пошук у ширину [212]:

```
# алгоритм пошуку в ширину
T = nx.bfs_tree(G, 1)

plt.figure(figsize=(6, 4))
nx.draw_networkx(T, with_labels=True)
```

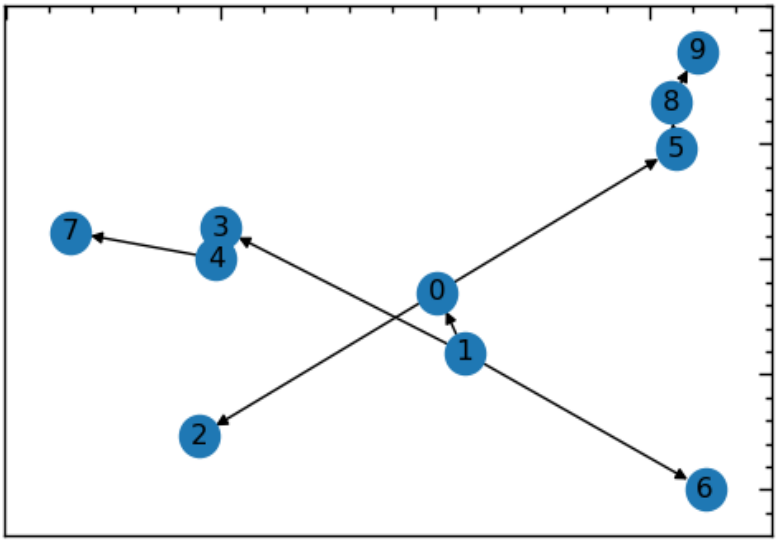


Рис. 13.20: Повертає орієнтоване дерево, побудоване на основі пошуку в ширину, починаючи з джерела

```
# усі найкоротші шляхи
nx.shortest_path_length(G, 1) # виводимо словник
```

```
{1: 0, 0: 1, 3: 1, 6: 1, 2: 2, 5: 2, 4: 2, 8: 3, 7: 3, 9: 4}
```

```
# середній найкоротший шлях
nx.average_shortest_path_length(G)
```

```
2.4
```

13.1.4.3.2 Зв'язні компоненти

У простій мережі для *кожної* пари вершин можна знайти шлях, який їх з'єднує. Це і є визначенням **зв'язного графа**. Ми можемо перевірити цю властивість для заданого графа:

```
nx.is_connected(G)
```

```
True
```

Не кожен граф зв’язний:

```
G_test = nx.Graph()

nx.add_cycle(G_test, (1,2,3))
G_test.add_edge(4, 5)

plt.figure(figsize=(6, 4))
nx.draw_networkx(G_test, with_labels=True)
```

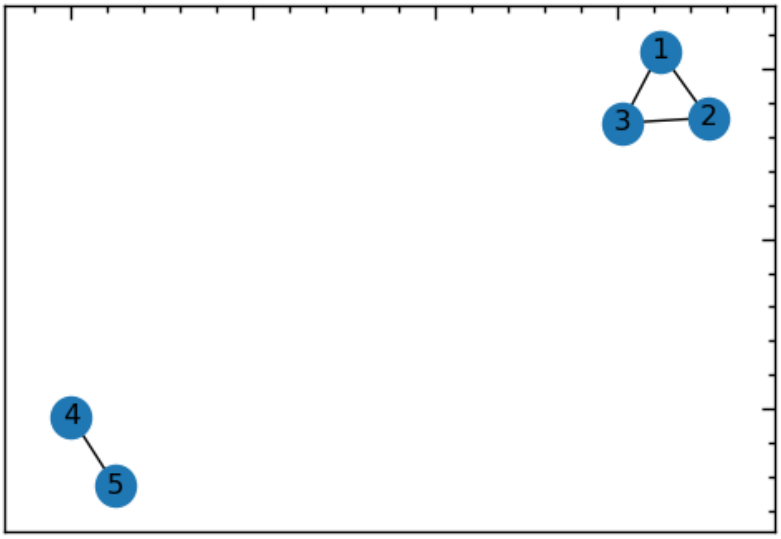


Рис. 13.21: Розв’язний граф із циклічним шляхом

```
nx.is_connected(G_test)
```

```
False
```

А NetworkX видасть помилку, якщо ви запитаєте шлях між вузлами, якого не існує:

```
nx.has_path(G_test, 3, 5)
```

```
False
```

```
nx.shortest_path(G_test, 3, 5)
```

```
NetworkXNoPath: No path between 3 and 5.
```

Візуально ми можемо ідентифікувати дві пов’язані компоненти на графі. Давайте перевіримо це:

```
nx.number_connected_components(G_test)
```

```
2
```

Функція `nx.connected_components()` отримує граф і повертає список наборів імен вершин, по одному такому набору для кожної зв’язної компоненти.

Перевірте, чи відповідають дві множини у наступному списку двом зв’язним компонентам на рисунку графа вище:

```
list(nx.connected_components(G_test))
```

```
[{1, 2, 3}, {4, 5}]
```

Якщо ви не знайомі з множинами у Python, це колекції елементів без дублікатів. Вони корисні для збору імен вузлів, оскільки імена вузлів повинні бути унікальними. Як і у випадку з іншими колекціями, ми можемо отримати кількість елементів у множині за допомогою функції `len`:

```
components = list(nx.connected_components(G_test))
len(components[0])
```

```
3
```

Нас часто цікавить найбільша зв’язна компонента, яку іноді називають *ядром* мережі. Ми можемо скористатися вбудованою функцією `max` у Python, щоб отримати найбільший зв’язну компоненту. За замовчуванням функція `max` у Python сортує дані у лексикографічному (тобто алфавітному) порядку, що не є корисним у даному випадку. Ми хочемо отримати максимальний зв’язаний компонент при сортуванні в порядку його розміру, тому ми передаємо `len` як ключову функцію:

```
max(nx.connected_components(G_test), key=len)
```

```
{1, 2, 3}
```

Хоча часто достатньо мати лише список назв вершин, іноді нам потрібен власне підграф, що містить найбільш зв’язну вершину. Один із способів отримати її — передати список назв вершин у функцію `G.subgraph()`:

```
core_nodes = max(nx.connected_components(G_test), key=len)
core = G.subgraph(core_nodes)

plt.figure(figsize=(6, 4))
nx.draw_networkx(core, with_labels=True)
```

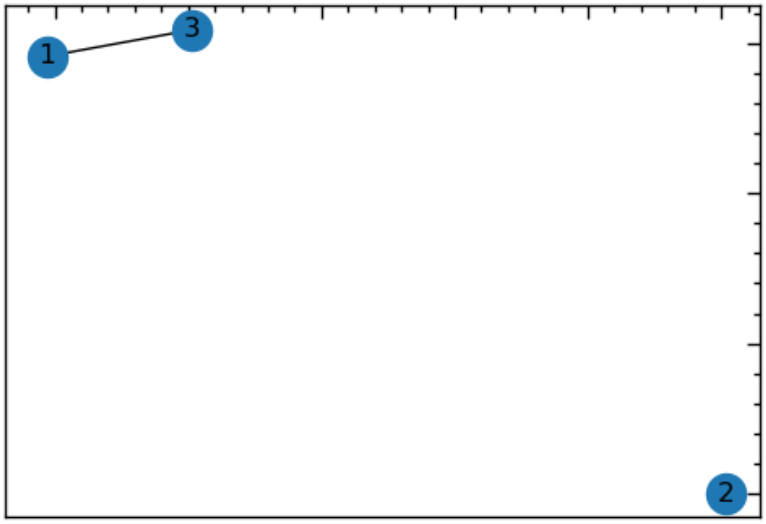


Рис. 13.22: Граф, що представляє підмножину з найбільш зв’язних компонент

Ті з вас, хто використовує завершення написання коду за допомогою табуляції, також помітять функцію `nx.connected_component_subgraphs()`. Її також можна використати для отримання основного підграфа, але представлений метод є більш ефективним, якщо вас цікавить найбільша зв’язна компонента.

13.1.4.3.3 Направлені шляхи та компоненти

Давайте поширимо ці ідеї про шляхи та зв’язні компоненти на орієнтовані графи.

```
D = nx.DiGraph()
D.add_edges_from([(1,2), (2,3), (3,2), (3,4), (3,5), (4,2), (4,5), (4,6), (5,6),
(6,4)])


plt.figure(figsize=(6, 4))
nx.draw_networkx(D, with_labels=True)
```

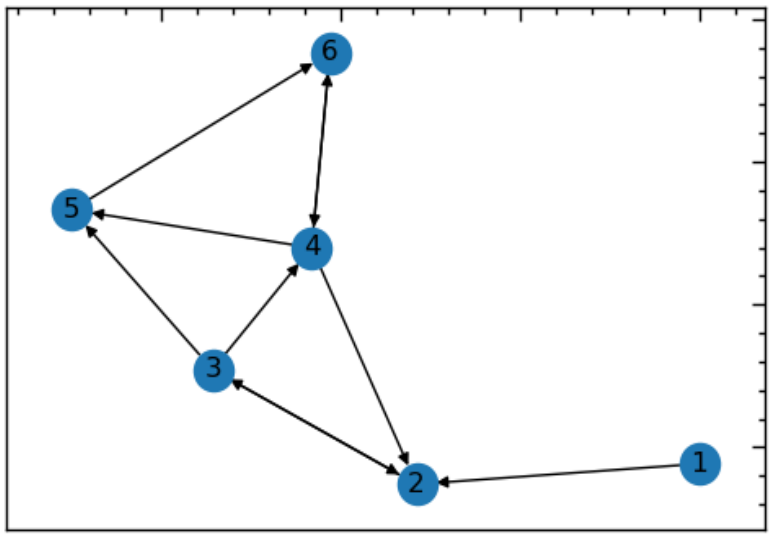


Рис. 13.23: Простий орієнтовний граф

Ми знаємо, що в орієнтованому графі ребро з довільної вершини u до довільної вершини v не говорить про те, що існує ребро з v до u. Тобто, для направленого графа ми спостерігатимемо асиметрію шляхів. Зверніть увагу, що цей граф має шлях від 1 до 4, але не у зворотному напрямку.

```
nx.has_path(D, 1, 4)
```

```
True
```

```
nx.has_path(D, 4, 1)
```

```
False
```

Інші функції NetworkX, що працюють зі шляхами, також враховують цю асиметрію:

```
nx.shortest_path(D, 2, 5)
```

```
[2, 3, 5]
```

```
nx.shortest_path(D, 5, 2)
```

```
[5, 6, 4, 2]
```

Оскільки немає ребра з 5 в 3, найкоротший шлях з 5 в 2 не може просто пройти назад по найкоротшому шляху з 2 в 5 — він повинен пройти довшим шляхом через вузли 6 і 4.

Направлені мережі мають два типи зв'язності. *Сильно зв'язні* означають, що між кожною парою вузлів існує спрямований шлях, тобто з будь-якого вузла ми можемо дістатися до будь-якого іншого вузла, дотримуючись спрямованості ребер. Уявіть собі автомобілі на мережі вулиць з одностороннім рухом: вони не можуть їхати проти потоку транспорту.

```
nx.is_strongly_connected(D)
```

```
False
```

Слабка зв'язність говорить про те, що між кожною парою вузлів існує шлях, незалежно від напрямку. Подумайте про пішоходів у мережі вулиць з одностороннім рухом: вони ходять по тротуарах, тому їх не хвилює напрямок руху.

```
nx.is_weakly_connected(D)
```

```
True
```

Якщо мережа сильно зв'язана, вона також є і слабко зв'язаною. Зворотне не завжди вірно, як видно з цього прикладу.

Функція `is_connected` для неорієнтованих графів видасть помилку, якщо задано орієнтований граф.

```
# Це призведе до помилки
nx.is_connected(D)
```

```
NetworkXNotImplemented: not implemented for directed type
```

У випадку направленого графа замість `nx.connected_components` тепер маємо `nx.weak_connected_components` та `nx.strong_connected_components`:

```
list(nx.weakly_connected_components(D))
```

```
[{1, 2, 3, 4, 5, 6}]
```

```
list(nx.strongly_connected_components(D))
```

```
[{2, 3, 4, 5, 6}, {1}]
```

13.1.4.4 Ексцентриситет

Ексцентриситет (eccentricity) вершини u в мережі — це максимальна відстань між u та будь-якою іншою вершиною мережі. Іншими словами, це максимальна довжина найкоротшого шляху між u та будь-якою іншою вершиною. Ексцентриситет мережі — це максимальний ексцентриситет будь-якого вузла мережі.

```
# ексцентриситет
# найбільша відстань між n та всіма іншими вершинами:
nx.eccentricity(G)
```

```
{0: 3, 1: 4, 2: 4, 3: 4, 4: 3, 5: 3, 6: 5, 7: 4, 8: 4, 9: 5}
```

```
# діаметр: max Ексцентриситет між двома вузлами у всій мережі (max max)
nx.diameter(G)
```

```
5
```

```
# Діаметр - максимальний ексцентриситет
max(nx.eccentricity(G).values())
```

```
5
```

```
# радіус: min Ексцентриситет між двома вузлами у всій мережі (min max)
nx.radius(G)
```

```
3
```

```
# радіус - мінімальний ексцентриситет
min(nx.eccentricity(G).values())
```

```
3
```

```
# периферія
# Ексцентриситет=діаметр
nx.periphery(G)
```

```
[6, 9]
```

```
# центр графа: Ексцентриситет = радіус
nx.center(G)
```

```
[0, 4, 5]
```

13.1.4.5 Центральність

Незалежно від того, чи представляють вузли людей, місця, комп'ютери або атоми, розташування вузла в структурі мережі тісно пов'язане з роллю, яку він відіграє в загальній системі. Різні структури уможливлюють різні ролі. Отже, кількісно оцінюючи структурні властивості вузла, можна зрозуміти роль, яку відіграє цей вузол. Числові міри, які характеризують мережні властивості вузла, називаються мірами **центральності** (centrality). Центральність часто вводять як міру важливості, але є багато способів, у які вузол може бути важливим. Наприклад, однією з найпростіших мір центральності є **ступенева центральність** (degree centrality). Ступенева центральність вузла — це просто кількість сусідів, яких він має (у спрямованій мережі є як ступеневі, так і неступеневі сусіди). У соціальній мережі ступенева центральність є мірою популярності.

13.1.4.5.1 Ступенева центральність — ненаправлені графи

Ступенева центральність — це міра важливості вузла в мережі, що базується на кількості зв'язків, які він має з іншими вузлами. Ступеневу центральність вершини i можна обчислити як $C_D(i) = k_i/(n-1)$, де k_i — ступінь вершини i, тобто кількість ребер, інцидентних вершині, а n — загальна кількість вершин у мережі. Знаменник $n-1$ використовується для того, щоб врахувати той факт, що вершина не може бути з'єднана сама з собою.

Ступенева центральність вузла коливається від 0 до 1, причому більше значення вказує на те, що вузол є більш центральним у мережі. Вузли з високою ступеневою центральністю, як правило, добре пов'язані з іншими вузлами, і їх видалення з мережі може мати значний вплив на її зв'язність.

Розглянемо деякі показники на прикладі графа **карате-клубу**.

```
# Карате-клуб
G_karate = nx.karate_club_graph()
G_karate = nx.convert_node_labels_to_integers(G_karate, first_label=1)

plt.figure(figsize=(6, 4))

# Встановіть положення вузлів за допомогою конструктора Камада-Каваї
pos = nx.kamada_kawai_layout(G_karate)
```

```
# Будуємо граф з червоними вузлами для вузла 0 (інструктор клубу) і вузла 33
(член клубу): тепер це 1 і 34.
red_nodes = [1, 34]
node_colors = ['red' if node in red_nodes else 'blue' for node in
G_karate.nodes()]
nx.draw_networkx_nodes(G_karate, pos, node_color=node_colors)
nx.draw_networkx_edges(G_karate, pos)

# Будуємо мітки для вузлів
nx.draw_networkx_labels(G_karate, pos);
```

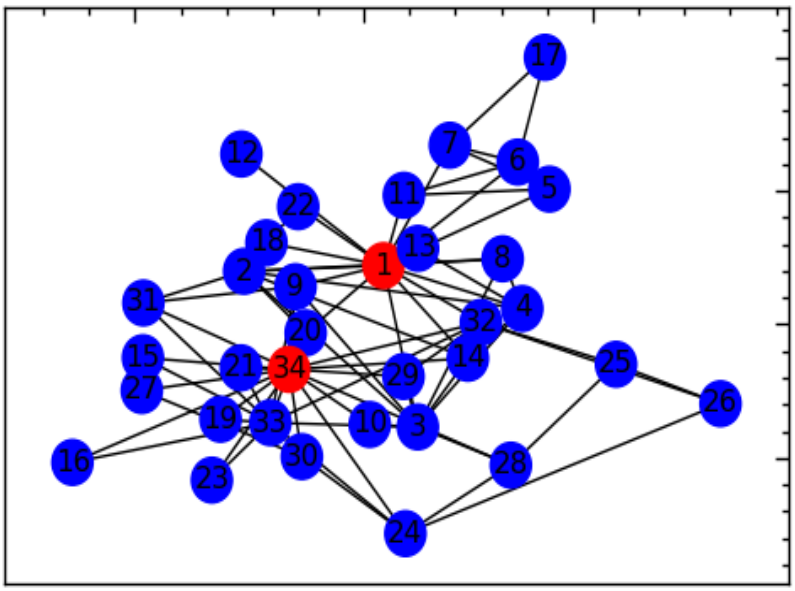


Рис. 13.24: Граф карате-клубу з виокремлини лідерами двох фракцій

```
# ступеневі центральності
degCent = nx.degree_centrality(G_karate)

# сортування за ступеневою центральністю
sorted_degcent = {k: v for k, v in sorted(degCent.items(), key=lambda item:
item[1], reverse=True)}

# ступенева центральність вузла

degCent[34]
```

```
0.5151515151515151
```

```
# відобразити мережу з розмірами вершин на основі їх ступеневої центральності
plt.figure(figsize=(6, 4))

# створити список розмірів вершин на основі ступеневої центральності
node_sizes = [10000*v*v for v in degCent.values()]

# будуємо граф
nx.draw_networkx(G_karate, with_labels=True, node_size=node_sizes,
pos=nx.spring_layout(G_karate))
```

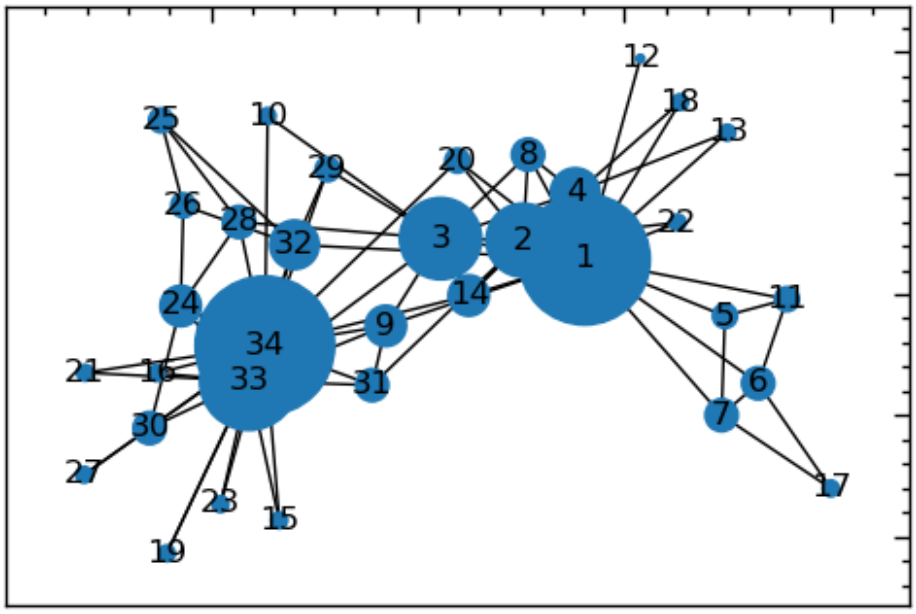


Рис. 13.25: Граф карате-клубу зі збільшеними вершинами на основі їх ступеневої центральності

```
# кольори на основі ступеневої центральності
node_colors = [v for v in degCent.values()]

plt.figure(figsize=(6, 4))
# будуємо граф
nx.draw_networkx(G_karate,
                 with_labels=True,
                 node_size=node_sizes,
                 pos=nx.spring_layout(G_karate),
                 node_color=node_colors,
                 cmap=plt.cm.PuBu)

# PuBu розшифровується як "Pu" (фіолетовий) - "Bu" (синій),
# і це послідовна карта кольорів, яка варіюється від світло-фіолетового до
темно-синього.
```

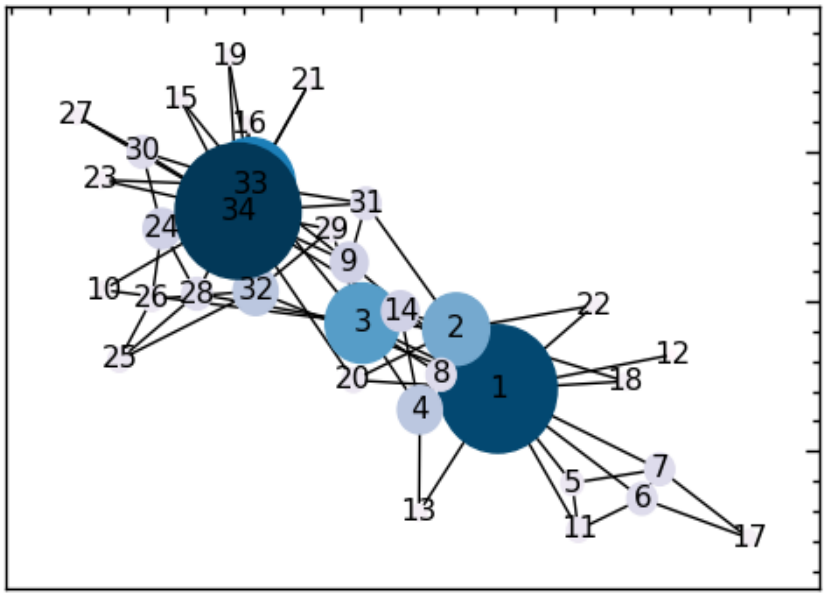


Рис. 13.26: Граф карате-клубу з виокремленими вершинами на основі їх ступеневої центральності за допомогою різної палітри кольорів

13.1.4.5.2 Ступенева центральність — направлені графи

```
# направлений граф
G = nx.DiGraph()

G.add_edge("A","B")
G.add_edge("A","D")
G.add_edge("A","C")
G.add_edge("B","D")

plt.figure(figsize=(6, 4))
# будуємо вузли з мітками
nx.draw_networkx(G, with_labels=True)
```

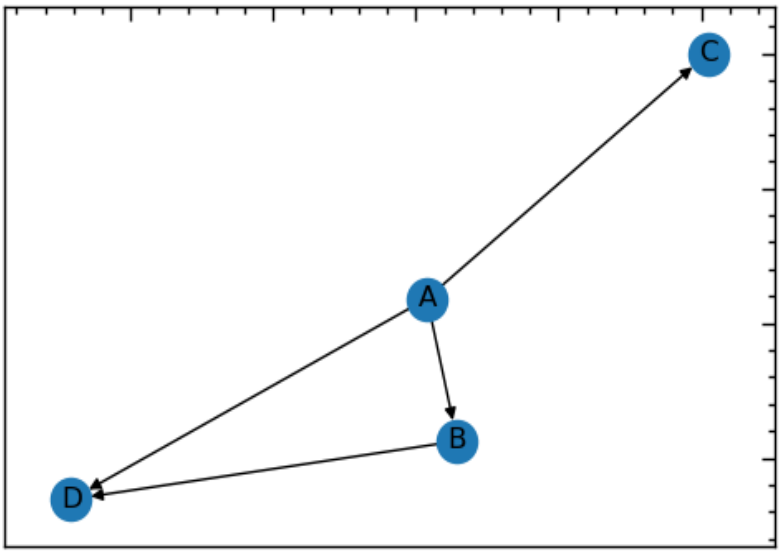


Рис. 13.27: Приклад направленого графа

```
# вхідний ступінь вершини
indegCent = nx.in_degree_centrality(G)
indegCent
```

```
{'A': 0.0,'B': 0.3333333333333333, 'D': 0.6666666666666666, 'C':
0.3333333333333333}
```

```
# вихідний
outdegCent = nx.out_degree_centrality(G)
outdegCent
```

```
{'A': 1.0, 'B': 0.3333333333333333, 'D': 0.0, 'C': 0.0}
```

```
# конкретна вершина
outdegCent["A"]
```

```
1.0
```

13.1.4.5.3 Ступінь близькості

Міра, відома як **ступінь близькості** (closeness centrality), є однією з найстаріших мір центральності, що використовується в мережній науці, запропонована соціологом Алексом Бавеласом у 1950 році. Близькість визначається як зворотна величина до **віддаленості** (farness). Що таке віддаленість? Більш зрозуміло, віддаленість вузла — це сума відстаней між цим вузлом і всіма іншими вузлами. Отже, вузол з високою центральністю близькості знаходиться буквально поруч з іншими вузлами. Центральність вузла вимірює, наскільки швидко він може поширювати інформацію або вплив по всій мережі, оскільки вузли з меншою середньою відстанню до всіх інших вузлів можуть спілкуватися більш ефективно. Крім того, вузли з високим показником центральності часто розташовані в центрі мережі, і їх видалення може мати значний вплив на зв'язність мережі.

Ступінь близькості вузла i можна обчислити як $C_C(i) = \left(\sum_{j \neq i} d_{ij}\right)^{-1}$, де d_{ij} — найкоротша відстань між вузлами i та j. Ступінь близькості вузла коливається від 0 до 1, причому більше значення вказує на меншу середню відстань до всіх інших вузлів мережі.

У наступному прикладі використовується функція NetworkX `closeness_centrality()` для обчислення значень центральності для мережі карате клубу та відображення 10 найближчих один до одного каратистів:

```
closeness = nx.closeness_centrality(G_karate)
sorted(closeness.items(), key=lambda x:x[1], reverse=True)[:10]
```

```
[(1, 0.5689655172413793), (3, 0.559322033898305), (34, 0.55), (32,
0.5409836065573771), (9, 0.515625), (14, 0.515625), (33, 0.515625), (20, 0.5),
(2, 0.4852941176470588), (4, 0.4647887323943662)]

# намалювати мережу з розмірами вершин на основі їх ступеня близькості

# створити список розмірів вершин на основі ступеня близькості
node_sizes = [3000*v*v for v in closeness.values()]

# кольори на основі ступеневої близькості
node_colors = [v for v in closeness.values()]

# будуємо граф
plt.figure(figsize=(6, 4))
nx.draw_networkx(G_karate,
                 with_labels=True,
                 node_size=node_sizes,
                 pos=nx.spring_layout(G_karate),
                 node_color=node_colors,
                 cmap=plt.get_cmap('plasma'))
```

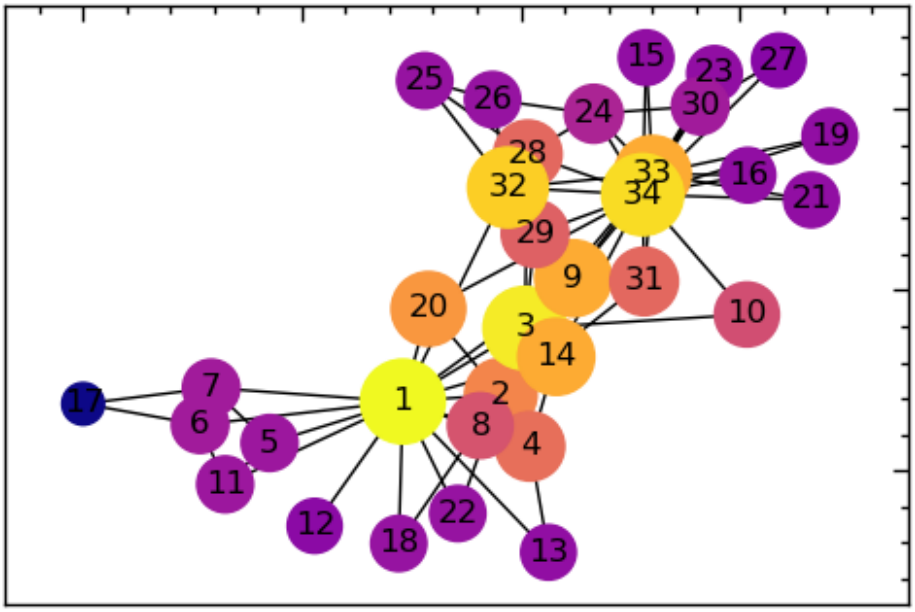


Рис. 13.28: Граф карате-клубу з виокремлини вершинами на основі їх ступеня близькості

13.1.4.5.4 Ступінь посередництва

У популярній дитячій грі “Телефон” один гравець починає з того, що шепоче повідомлення іншому, той шепоче це повідомлення іншому і так далі. Врешті-решт, останній гравець промовляє повідомлення вголос. Як правило, фінальне повідомлення не має нічого спільного з початковим. Кожного разу, коли повідомлення передається від людини до людини, воно може змінюватися, можливо, через те, що його неправильно почули, а можливо, через те, що його

навмисно змінили. У більш складних соціальних мережах, таких як організації та громадські рухи, особи, які з'єднують різні частини мережі, мають найбільші можливості фільтрувати, посилювати та змінювати інформацію. Таких людей називають брокерами, а ребра, що з'єднують віддалені частини мережі, — мостами. Важливість таких вузлів і ребер не обмежується соціальними мережами. У потокових мережах — таких як залізниці, водопроводи та телекомунікаційні системи — вузли що з'єднують віддалені частини мережі, можуть діяти як вузькі місця, обмежуючи обсяг потоку. Виявлення таких вузьких місць дає змогу збільшити їхню пропускну здатність і захистити їх від збоїв та атак. Мости і брокери важливі, тому що вони знаходяться між різними частинами мережі. Відповідно, тип центральності, який використовується для визначення мостів і брокерів називається **ступенем посередництва** (betweenness centrality).

Ступінь посередництва — це міра того, наскільки вузол лежить на найкоротших шляхах між іншими вузлами мережі. Ступінь посередництва для вузла i можна обчислити як $C_B(i) = \sum_{s \neq i \neq t} \sigma_{st}(i)/\sigma_{st}$, в якій s і t — два вузли мережі, σ_{st} — загальна кількість найкоротших шляхів між s і t, а $\sigma_{st}(i)$ — кількість найкоротших шляхів між s і t, які проходять через вузол i.

Ступінь посередництва змінюється від 0 до 1, причому більше значення вказує на більшу кількість найкоротших шляхів, що проходять через вершину. Вузли з високим значенням центральності часто розташовані на "мостах" між різними кластерами або спільнотами в мережі, і їх видалення може мати значний вплив на зв'язність мережі.

Ступінь посередництва базується на припущенні, що чим більше найкоротших шляхів проходить через вершину (або ребро), тим більше вона виступає в ролі брокера (або моста). Для ступеня посередництва знаходять найкоротші шляхи між кожною парою вузлів. Значення ступеня посередництва для вузла або ребра — це просто кількість цих шляхів, що проходять через нього. На наступній діаграмі показано приклад мережі та розраховані значення посередництва для кожної вершини та ребра. Для кожної пари вершин показано найкоротший шлях (за винятком тривіальних шляхів довжиною 1). Посередництво вузла — це сума шляхів, які проходять через цей вузол. Посередництво ребра — це кількість нетривіальних шляхів, які проходять через це ребро, плюс 1 для самого ребра.

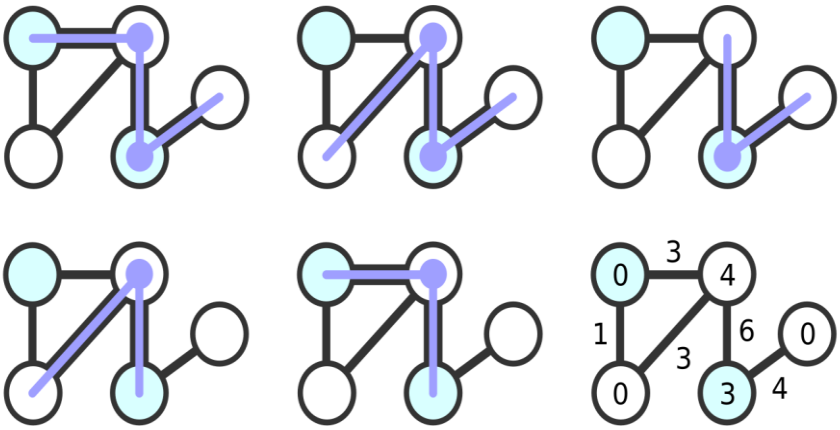


Рис. 13.29: Усі нетривіальні найкоротші шляхи та отримані центри посередництва

Ступінь посередництва між вузлами легко обчислюється в NetworkX за допомогою функції `betweenness_centrality()`. Ця функція повертає словник, який зіставляє позначення вузлів зі значеннями посередництва. Якщо аргумент `normalized` має значення `True` (за замовчуванням), значення ступеня посередництва ділиться на кількість пар вузлів, що може бути корисним для порівняння значень посередництва, що мають різні масштаби. Якщо аргумент `endpoints` має значення `True` (за замовчуванням False), то кінцеві точки шляху будуть включені в розрахунок посередництва.

```
btwnCent = nx.betweenness_centrality(G_karate, endpoints=False)
sorted(btwnCent.items(), key=lambda x: x[1], reverse=True)[:10]
```

```
[(1, 0.43763528138528146),(34, 0.30407497594997596), (33, 0.145247113997114),
(3, 0.14365680615680618), (32, 0.13827561327561325), (9, 0.05592682780182781),
(2, 0.053936688311688304), (14, 0.04586339586339586), (20, 0.03247504810004811),
(6, 0.02998737373737374)]
```

```
# відобразити мережу з розмірами вершин на основі їх ступеня посередництва

# створити список розмірів вершин на основі ступеня посередництва
node_sizes = [10000*v*v for v in btwnCent.values()]

# кольори на основі ступеня посередництва
node_colors = [v for v in btwnCent.values()]

# будуємо граф
plt.figure(figsize=(6, 4))
nx.draw_networkx(G_karate, with_labels=True,
        node_size=node_sizes,
        pos=nx.spring_layout(G_karate),
        node_color=node_colors,
        cmap=plt.get_cmap('viridis'))
```

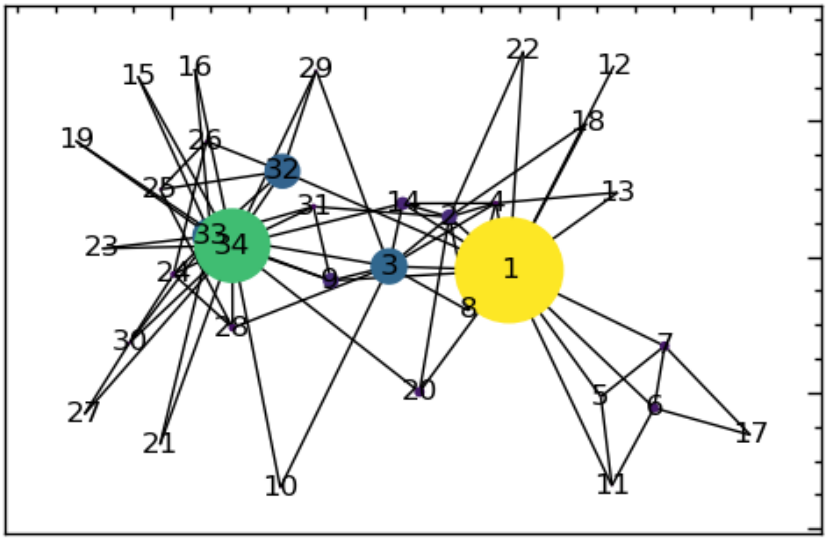


Рис. 13.30: Граф карате-клубу з виокремлини вершинами на основі їх ступеня посередництва вершин

Видно, що високим рівнем посередництва характеризуються вершини 1, 34 і 33. Високий рівень посередництва між ними свідчить про те, що ці особи є важливими інформаційними посередниками в клубі карате. Можливо, вони є найбільш вправними каратистами.

Ступінь посередництва для ребер — це міра того, наскільки ребро лежить на найкоротших шляхах між іншими ребрами в мережі. Посередництво ребра e можна обчислити так:

$$C_B(e) = \sum_{s \neq e \neq t} \sigma_{st}(e)/\sigma_{st}.$$

У рівнянні вище s і t — дві вершини мережі, σ_{st} — загальна кількість найкоротших шляхів між s і t, а $\sigma_{st}(e)$ — кількість найкоротших шляхів між s і t, які проходять через ребро e.

Ступінь посередництва ребра змінюється від 0 до 1, причому більше значення вказує на більшу кількість найкоротших шляхів, які проходять через ребро. Ребра з високою посередництвом часто розташовані на “мостах” між різними кластерами або спільнотами в мережі, і їх видалення може мати значний вплив на зв’язність мережі.

```
btwnCent_edge = nx.edge_betweenness_centrality(G_karate, normalized=True)
sorted(btwnCent_edge.items(), key=lambda x: x[1], reverse=True)[:10]
```

```
[((1, 32), 0.1272599949070537), ((1, 7), 0.07813428401663695), ((1, 6),
0.07813428401663694), ((1, 3), 0.0777876807288572), ((1, 9),
0.07423959482783014), ((3, 33), 0.06898678663384543), ((14, 34),
```

```
0.06782389723566191), ((20, 34), 0.05938233879410351), ((1, 12),
0.058823529411764705), ((27, 34), 0.0542908072319837)]

# візуалізувати мережу з розмірами вершин на основі ступеня посередництва їх
ребер

# кольори ребер на основі ступеня посередництва ребер
edge_colors = [v for v in btwnCent_edge.values()]
edge_widths = [v*100 for v in btwnCent_edge.values()]

# будуємо граф
plt.figure(figsize=(6, 4))
nx.draw_networkx(G_karate,
        with_labels=True,
        pos=nx.spring_layout(G_karate),
        edge_color=edge_colors,
        width=edge_widths)
```

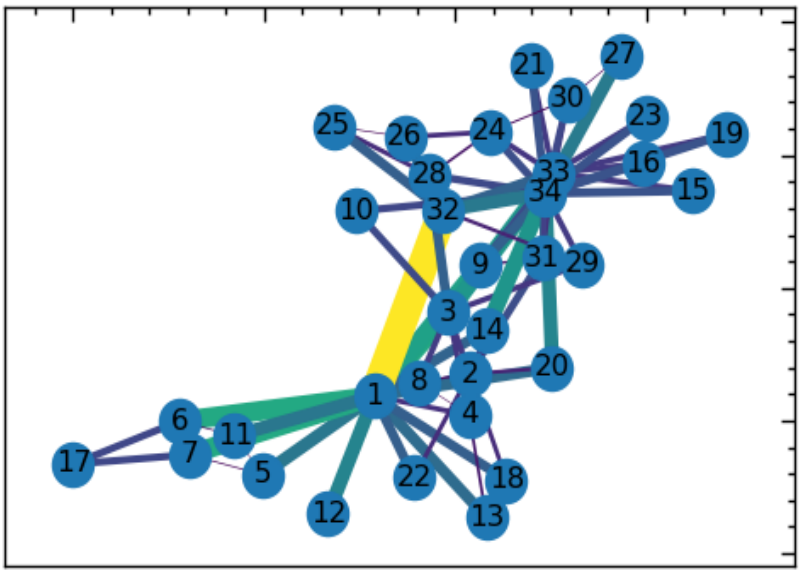


Рис. 13.31: Граф карате-клубу з виокремлини ребрами на основі їх ступеня посередництва ребер

Якщо розглядати, наприклад, топ 3 ребер із найбільшим ступенем посередництва, ми побачимо, що, як правило, найкраща комунікація проходить у тренера з учнями під номерами 32, 7, 6, 3 тощо.

13.1.4.5.5 Ступінь впливовості

Уявіть, що у вас є важливе повідомлення, яке потрібно донести до цілої групи (наприклад, до вашого роботодавця або школи), але ви можете передати його лише одній людині. Кому б ви це сказали? Ви б хотіли знайти когось, хто має хороші зв’язки з усією мережею. Ви можете спробувати звернутися до людини з найвищою ступеневою центральністю (найбільшою кількістю друзів). Недоліком такого підходу є те, що її друзі можуть бути не дуже добре пов’язані

з рештою мережі. Наприклад, у гіпотетичній компанії директор з продажу в окремому регіоні може знати найбільше людей, але не знати, як зв'язатися з іншими відділами чи регіонами. Замість нього краще знайти когось, хто має тісні зв'язки з іншими людьми, які мають тісні зв'язки, наприклад, генерального директора (або, що більш ймовірно, його помічника). Таких людей іноді називають **хабами**, тому що, подібно до центру колеса зі спицями, вони з'єднують між собою багато різних точок. Цю концепцію високозв'язних хабів добре відображає показник, який називається **ступенем впливовості** (eigenvector centrality).

Ступінь впливовості вершини i можна визначити через головний власний вектор матриці суміжності **A** мережі:

$$\mathbf{Av} = \lambda \mathbf{v},$$

де **v** — власний вектор, що відповідає найбільшому власному значенню λ. Ступінь впливовості вершини i задається i-им елементом **v**.

Ступінь впливовості вузла коливається від 0 до 1, причому більше значення вказує на більшу важливість вузла та його сусідів у мережі. Вузли з високим ступенем впливовості часто розташовані в центрі мережі і добре пов'язані з іншими сильно пов'язаними вузлами, і їх видалення може мати значний вплив на зв'язність мережі.

```
eigenvector_centrality = nx.eigenvector_centrality_numpy(G_karate)
sorted(eigenvector_centrality.items(), key=lambda x: x[1], reverse=True)[:10]
```

```
[(34, 0.3733634702914831), (1, 0.3554914445245666), (3, 0.3171925044864317),
(33, 0.30864421979104706), (2, 0.2659599195524917), (9, 0.22740390712540018),
(14, 0.22647272014248135), (4, 0.21117972037789046), (32, 0.19103384140654373),
(31, 0.17475830231435288)]
```

```
# відображаємо мережу з розмірами вершин на основі їх ступеня впливовості

# створити список розмірів вершин на основі ступеня впливовості
node_sizes = [10000*v*v for v in eigenvector_centrality.values()]

# кольори на основі ступеневої впливовості
node_colors = [v for v in eigenvector_centrality.values()]

# будуємо граф
plt.figure(figsize=(6, 4))
nx.draw_networkx(G_karate,
        with_labels=True,
        node_size=node_sizes,
        pos=nx.spring_layout(G_karate),
        node_color=node_colors,
        cmap=plt.get_cmap('Purples'))
```

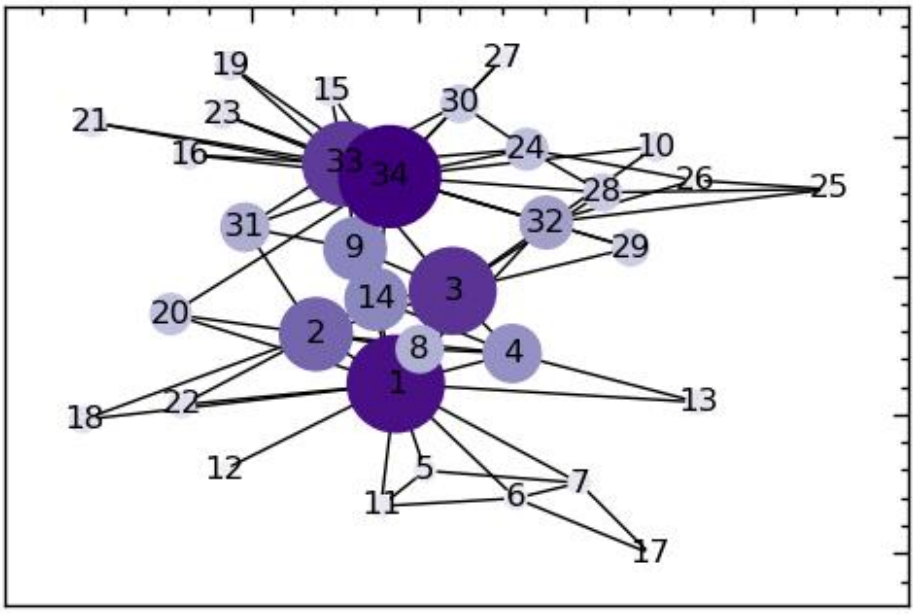


Рис. 13.32: Граф карате-клубу з виокремлини вершини на основі їх ступеня впливовості

13.1.5 Широкомасштабний опис мереж

Широкомасштабні структури можуть сильно відрізнятися від мережі до мережі. Ці відмінності часто вказують на різні типи мереж (наприклад, соціальні та технологічні). Широкомасштабні структури також можуть мати важливі наслідки для функціональних властивостей, таких як як стійкість до збоїв і атак. Розглянемо аналіз структурних показників для мереж різних типів.

Як ви вже могли переконатися на прикладі графа карате-клубу, NetworkX надає декілька вбудованих наборів мережних даних, які можна використовувати для тестування та експериментів. Ці набори даних доступні в самій бібліотеці NetworkX і можуть бути завантажені за допомогою функцій, які починаються з префікса `nx.`, за яким слідує назва набору даних.

Ось кілька прикладів вбудованих мережних наборів даних у NetworkX:

- nx.karate_club_graph() — повертає мережу Zachary's Karate Club, соціальну мережу карате-клубу, де кожен вузол представляє члена клубу, а кожне ребро представляє дружні стосунки між членами;
- nx.les_miserables_graph() — повертає мережу персонажів роману Віктора Гюго "Знедолені", де кожен вузол представляє персонажа роману, а кожне ребро представляє спільну появу двох персонажів у главі;
- nx.davis_southern_women_graph() — повертає мережу соціальних взаємодій між жінками у містечку на півдні США в 1930-х роках, де кожен

вузол представляє жінку, а кожне ребро — соціальні стосунки між двома жінками.

Це лише кілька прикладів вбудованих мережних наборів даних у NetworkX. Ви можете знайти більше інформації про доступні набори даних та їх використання в документації NetworkX.

```
# генеруємо першу мережу
G_karate = nx.karate_club_graph()
mr_hi = 0
john_a = 33

# генеруємо другу мережу
G_novel = nx.les_miserables_graph()

# генеруємо третю мережу
G_woman = nx.davis_southern_women_graph()
```

Наступний код візуалізує три приклади мереж:

```
fig, ax = plt.subplots(1, 3, figsize=(8, 6))

ax[0].set_title("Карате")
nx.draw_networkx(G_karate, node_size=0, with_labels=False, ax=ax[0])
ax[1].set_title("Роман")
nx.draw_networkx(G_novel, node_size=0, with_labels=False, ax=ax[1])
ax[2].set_title("Жінки")
nx.draw_networkx(G_woman, node_size=0, with_labels=False, ax=ax[2])

plt.tight_layout()
```

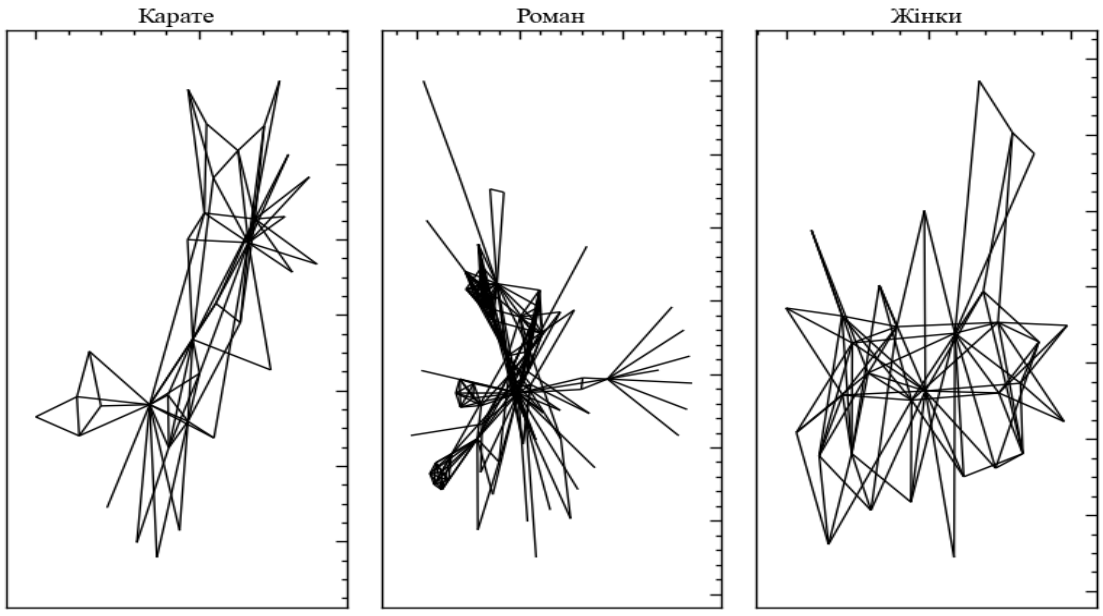


Рис. 13.33: Мережі карате-клубу, персонажів роману та соціальної взаємодії між жінками в містечку на півдні США в 1930-х роках

13.1.5.1 Діаметр і найкоротший шлях

Мережі можуть бути охарактеризовані відповідно до розподілу довжини найкоротшого шляху. Наведена нижче функція будує гістограму всіх найкоротших шляхів у мережі:

```
def path_length_histogram(G, title=None):
# знаходимо довжини шляхів
    length_source_target =dict(nx.shortest_path_length(G))
# конвертуємо словник словників до звичайного списку
    all_shortest = sum([
                   list(length_target.values())
                   for length_target
                   in length_source_target.values()],
                   [])
# розраховуємо цілочисельні біни
    high = max(all_shortest)
    bins = [-0.5+i for i in range(high+2)]
# будуємо гістограму
    plt.hist(all_shortest, bins=bins, rwidth=0.8)
    plt.title(title)
    plt.xlabel("Відстань")
    plt.ylabel("Підрахунок")
```

Тепер давайте порівняємо розподіл довжин шляхів для трьох мереж:

```
# Створюємо рисунок
plt.figure(figsize=(8, 5))
# Будуємо гістограми найкоротших шляхів
plt.subplot(1, 3, 1)
path_length_histogram(G_karate, title="Карате")
plt.subplot(1, 3, 2)
path_length_histogram(G_novel, title="Роман")
plt.subplot(1, 3, 3)
path_length_histogram(G_woman, title="Жінки")

plt.tight_layout()
```

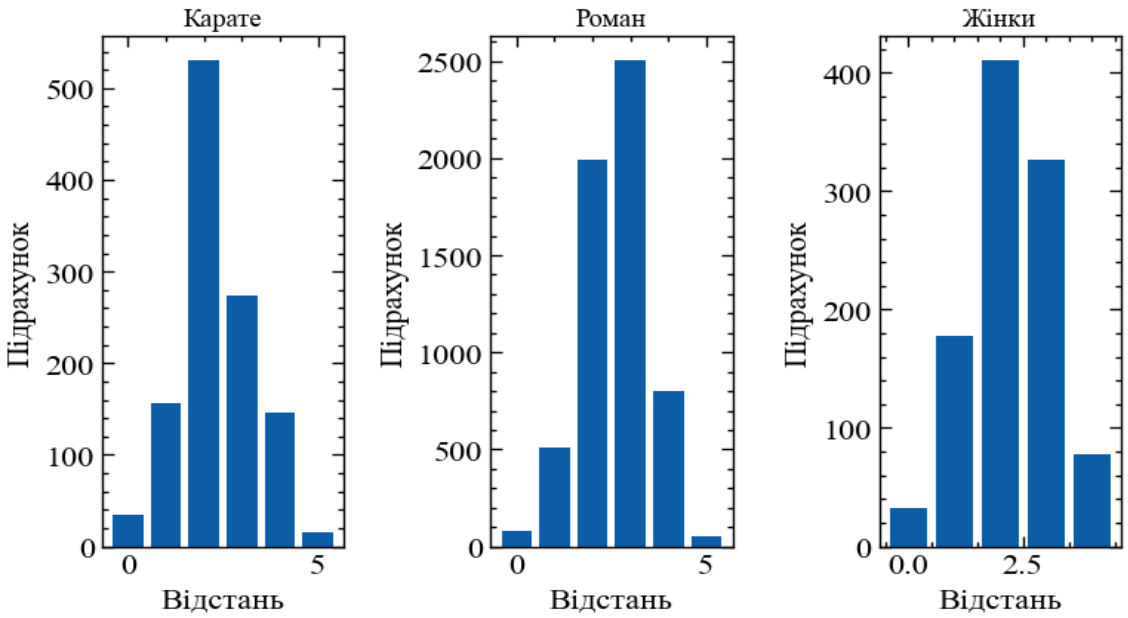


Рис. 13.34: Гістограми найкоротших шляхів у мережах карате-клубу, персонажів роману та соціальної взаємодії між жінками в містечку на півдні США в 1930-х роках

Усі три графи мають достатньо малі найкоротші шляхи. Соціальні мережі, як правило, мають короткі шляхи, що відомо як **феномен малого світу**.

Хоча розподіл повної довжини шляху є інформативним, він є дещо громіздким, тому корисно використовувати агреговані показники. Однією з таких мір є **середня довжина найкоротшого шляху**, яку можна обчислити наступним чином:

```
print("Середній найкоротший шлях для карате-клубу: ",
nx.average_shortest_path_length(G_karate))
print("Середній найкоротший шлях для роману: ",
nx.average_shortest_path_length(G_novel))
print("Середній найкоротший шлях для жінок: ",
nx.average_shortest_path_length(G_woman))
```

```
Середній найкоротший шлях для карате-клубу:  2.408199643493761
Середній найкоротший шлях для роману:  2.6411483253588517
Середній найкоротший шлях для жінок:  2.306451612903226
```

⚠ Попередження

У роз’єднаній на дві або або більше компонентів мережі без ребра між ними середня довжина шляху стає нескінченною. Цю проблему можна вирішити кількома способами, наприклад, використання гармонічного, а не арифметичного середнього, або усереднення середнього значення найкоротших шляхів у межах кожної зв’язної компоненти. Який метод є доречним, залежить від типу мережі, що аналізується

Крім того, розмір мережі може бути охарактеризований найбільшою довжиною шляху довжиною, яка називається **діаметром**. Діаметри трьох прикладів мереж можна знайти за допомогою функції `diameter()`:

```
print("Діаметр для карате-клубу: ", nx.diameter(G_karate))
print("Діаметр для роману: ", nx.diameter(G_novel))
print("Діаметр для жінок: ", nx.diameter(G_woman))
```

```
Діаметр для карате-клубу:  5
Діаметр для роману:  5
Діаметр для жінок:  4
```

Як ми можемо бачити результати доволі схожі на попередні. На відміну від середньої довжини найкоротшого шляху, діаметр залежить лише від одного шляху. Як наслідок, один викид може значно збільшити діаметр. Однак у такому разі діаметр може бути гарним показником найгіршої довжини шляху.

13.1.5.2 Вимірювання стійкості мережі

Стійкість (resilience) — це здатність системи протистояти збоям і атакам. Наприклад, в електромережі стійкість означає продовження подачі електроенергії, коли лінія електропередач або генератор вийшли з ладу. У дорожньому русі це може означати можливість перенаправляти автомобілі, коли вулиця перекрита через аварію.

Стійкість — це фундаментальна властивість мережі, оскільки вона зазвичай досягається за допомогою резервних шляхів. Коли один шлях більше не доступний, інші все ще можуть бути використані.

Найпростішим (і найгрубішим) показником стійкості є **щільність мережі** (density): частка можливих ребер, які існують. Чим більше ребер у мережі, тим більше надлишкових шляхів існує між її вузлами. Наступний код використовує функцію `density()` для обчислення цього значення:

```
print("Щільність для карате-клубу: ", nx.density(G_karate))
print("Щільність для роману: ", nx.density(G_novel))
print("Щільність для жінок: ", nx.density(G_woman))
```

```
Щільність для карате-клубу:  0.13903743315508021
Щільність для роману:  0.08680792891319207
Щільність для жінок:  0.17943548387096775
```

Мережа зазвичай вважається розрідженою, якщо кількість ребер близька до N (кількість вузлів), і щільною, якщо кількість ребер близька до N^2.

Можна бачити, що найбільш стійкою (щільною) серед усіх трьох графів є мережа жінок.

13.1.5.3 Найменші розрізи

Більш складні показники відмовостійкості базуються на концепції найменших розрізів. **Найменший розріз** (min-cut) — це кількість вузлів (або ребер), які потрібно видалити, щоб розділити мережу на дві незв'язані частини. Найменші розрізи можна знайти або між двома конкретними вузлами, або над усіма парами вузлів.

У NetworkX найменший розріз між двома вузлами знаходять за допомогою функції `minimum_st_node_cut()`. Зауважте, що ця функція знаходиться у пакеті `connectivity` і має бути імпортована окремо на додачу до базового пакету `networkx`. Наступний код знаходить мінімальну довжину шляху між містером Хі та Джоном А. у мережі карате-клубу:

```
import networkx.algorithms.connectivity as nxcon
nxcon.minimum_st_node_cut(G_karate, mr_hi, john_a)
```

```
{2, 8, 13, 19, 30, 31}
```

Попередній результат говорить про те, що вузли 2, 8, 12, 19, 30, 31 потрібно видалити, щоб розділити мережу на дві половини, одна з яких містить містера Хі, а інша — Джона А.

Аналогічно, найменший розріз ребер:

```
nxcon.minimum_st_edge_cut(G_karate, mr_hi, john_a)
```

```
{(0, 8), (0, 31), (1, 30), (2, 8), (2, 27), (2, 28), (2, 32), (9, 33), (13, 33),
(19, 33)}
```

Якщо потрібно знати лише розмір найменшого розрізу, можна скористатися функціями `node_connectivity()` або `edge_connectivity()` базового пакету `networkx`. У наступному прикладі обчислюються ці значення для мережі карате-клубу:

```
nx.node_connectivity(G_karate, mr_hi, john_a)
```

```
6
```

```
nx.edge_connectivity(G_karate, mr_hi, john_a)
```

```
10
```

13.1.5.4 Зв'язність

Найменші розрізи можуть бути використані для визначення показників **зв'язності** (connectivity) для всієї мережі. Ці міри дуже корисні для кількісної оцінки стійкості мережі.

Зв'язність вузлів — це найменший мінімальний розріз між усіма парами вузлів. Зв'язність ребер визначається аналогічно. Фактичні значення розрізів між вузлами та ребрами можна знайти за допомогою пакету `connection`:

```
nxcon.minimum_node_cut(G_karate)
```

```
{0}
```

```
nxcon.minimum_edge_cut(G_karate)
```

```
{(11, 0)}
```

Зв’язність можна обчислити за допомогою функцій `node_connectivity()` та `edge_connectivity()`, не вказуючи вихідні та цільові вузли. У наступному прикладі обчислюється зв’язність вузлів для трьох прикладів мереж:

```
nx.node_connectivity(G_karate)
```

```
1
```

```
nx.node_connectivity(G_novel)
```

```
1
```

```
nx.node_connectivity(G_woman)
```

```
2
```

Здається, що всі ці мережі, окрім мережі жінок, можна роз’єднати, видаливши лише один вузок. Для мережі жінок потребується видалити два вузли.

Попередня міра зв’язності знаходить розмір найменшого мінімального розрізу, але його видалення не вплине на всі шляхи в мережі. Після видалення вузла або ребра мережа буде розділена, але в кожній половині вузли все ще будуть з’єднані один з одним.

Кращий показник надійності можна знайти, усереднивши зв’язність по всіх вузлах або ребрах за допомогою функцій `average_node_connectivity()` і `average_edge_connectivity()`. Зауважте, що обчислення цих значень може зайняти багато часу, навіть для невеликих мереж. Наступний код обчислює середню зв’язність вузлів для досліджуваних мереж:

```
print("Середня зв'язність для карате-клубу: ",
nx.average_node_connectivity(G_karate))
print("Середня зв'язність для роману: ", nx.average_node_connectivity(G_novel))
print("Середня зв'язність для жінок: ", nx.average_node_connectivity(G_woman))
```

```
Середня зв'язність для карате-клубу:  2.2174688057040997
Середня зв'язність для роману:  2.2624743677375254
Середня зв'язність для жінок:  3.7399193548387095
```

Мережа каратистів та персонажів роману доволі подібні один до одного по зв’язності, але мережа жінок представляється найбільш стійкою або організованою.

13.1.5.5 Централізація та нерівномірність

Мережі також можна класифікувати за **ступенем централізації** (centrality) — наскільки вони зосереджені в одному або декількох вузлах. Нерівномірний розподіл є більш централізованим. Наприклад, найбільш централізованою мережею є мережа, всі вузли якої під'єднані до одного вузла-хабу. Наступний код будує гістограми ступенів впливовості для кожної з мереж:

```
# Функція для побудови гістограми
def centrality_histogram(x, title=None):
    plt.hist(x, density=True)
    plt.title(title)
    plt.xlabel("Впливовість")
    plt.ylabel("Підрахунок")

# Створення рисунку
plt.figure(figsize=(8, 5))
# Розрахунок центральностей для кожного графа
plt.subplot(1, 3, 1)
centrality_histogram(
nx.eigenvector_centrality(G_karate).values(), title="Карате")
plt.subplot(1, 3, 2)
centrality_histogram(
nx.eigenvector_centrality(G_novel).values(),
title="Роман")
plt.subplot(1, 3, 3)
centrality_histogram(
nx.eigenvector_centrality(G_woman).values(), title="Жінки")

plt.tight_layout()
```

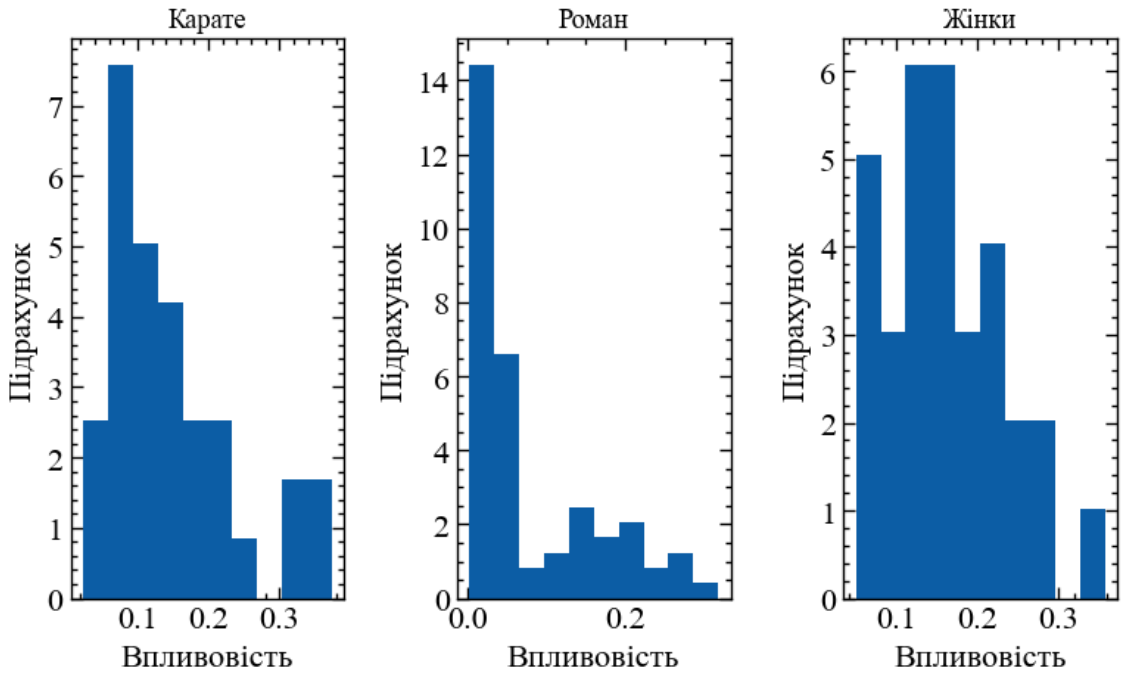


Рис. 13.35: Гістограми ступенів впливовості в мережах карате-клубу, персонажів роману та соціальної взаємодії між жінками в містечку на півдні США в 1930-х

роках

З представлених гістограм видно, що найвищі значення впливовості приходяться на мережу жінок. Найгіршою за впливовістю є мережа персонажів роману.

Виміряти нерівномірність набору значень можна за допомогою **ентропії Шеннона**. Чим більш рівномірно розподіленим є набір чисел, тим вища його ентропія. Наступна функція повертає ентропію списку чисел:

```
import math
def entropy(x):
# Нормалізація
    total = sum(x)
    x = [xi / total for xi in x]
    H =sum([-xi * math.log2(xi) for xi in x])
    return H
```

Обчислення ентропії ступенів впливовості у кожній з мереж дає наступний результат:

```
print("Ентропія ступенів впливовості для карате-клубу: ",
      entropy(nx.eigenvector_centrality(G_karate).values()))
print("Ентропія ступенів впливовості для роману: ",
      entropy(nx.eigenvector_centrality(G_novel).values()))
print("Ентропія ступенів впливовості для жінок: ",
      entropy(nx.eigenvector_centrality(G_woman).values()))
```

```
Ентропія ступенів впливовості для карате-клубу:  4.842401948329853
Ентропія ступенів впливовості для роману:  5.52075429881287
Ентропія ступенів впливовості для жінок:  4.858808158743919
```

Найбільш рівномірно розподіленою в даному випадку представляється мережа персонажів роману. Мережі карате-клубу та жінок мають трохи вищий ступінь централізації.

У соціальних мережах не всі стосунки є рівними. У соціології міцність стосунків вимірюється поняттям **міцність зв'язності** (tie strength). У цьому контексті зв'язність — це певний вид міжособистісних стосунків, а міцність — це будь-яка міра того, наскільки інтенсивними чи близькими є ці стосунки (зв'язності).

У 1973 році соціолог Марк Грановеттер описав важливість слабких зв'язків для зближення різних спільнот. Якщо всі зв'язки всередині спільноти сильні, то будь-які зв'язки між спільнотами мають бути слабкими. Він назвав це явище силою слабких зв'язків. З'єднуючи різні спільноти, слабкі зв'язки дають змогу знаходити інформацію з віддалених частин мережі. Але як виміряти силу зв'язностей?

13.1.5.6 Сила зв’язності

У мережі карате-клубу немає ніякої додаткової інформації про міцність ребер, але є відповідні властивості цих ребер, які можна обчислити, наприклад, **сила зв’язності**. Сила зв’язності зростає зі збільшенням кількості сусідів, які мають спільні вершини. Це мотивовано спостереженням, що близькі друзі, як правило, мають більше спільних друзів, і це часто може дати уявлення про структуру соціальної мережі. Наступний код обчислює силу зв’язку, використовуючи метод `neighbors()` для пошуку сусідів вузлів:

```
def tie_strength(G, v, w):
# Отримуємо сусідів вершин v та w у G
    v_neighbors = set(G.neighbors(v))
    w_neighbors = set(G.neighbors(w))
# Повернути розмір заданої зв'язності
    return 1+len(v_neighbors & w_neighbors)
```

Тут ми визначили міцність зв’язку як кількість спільних сусідів плюс один. Чому плюс один? Нульова вага умовно означає відсутність ребра, тому без додаткової одиниці ребра між вершинами, які не мають спільних сусідів, не вважатимуться ребрами.

```
G = nx.karate_club_graph()

# Надаємо інформацію про те, хто в якому клубі
# опинився після розділення клубу
member_club = [
0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1]

nx.set_node_attributes(G, dict(enumerate(member_club)), 'club')

# Знаходимо внутрішні та зовнішні ребра
for v, w in G.edges:
# Перебираємо пари вершин
# Встановлюємо 'True', якщо вершини в одному кластері (клубі)
   if G.nodes[v]["club"] == G.nodes[w]["club"]:
      G.edges[v, w]["internal"] = True
   else:
      G.edges[v, w]["internal"] = False

# Внутріші - каратисти знаходяться в одному клубі й підтримують зв'язок
internal = [e for e in G.edges if G.edges[e]["internal"]]

# Зовнішні - каратисти в різних клубах, але продовжують підтримувати зв'язок
external = [e for e in G.edges if ~G.edges[e]["internal"]]
```

Наступний код обчислює силу зв’язності кожного ребра і зберігає її в змінну `strength`:

```
strength = dict(((v,w), tie_strength(G, v, w)) for v, w in G.edges())
```

13.1.5.7 Мостовий проліт

Міцність зв’язків також можна оцінити кількісно, розглядаючи ефект видалення ребра з мережі. Вузли, з’єднані ребром, завжди знаходяться на відстані в 1 крок один від одного (у незваженій мережі). Але, якщо це ребро видалити, його кінцеві точки можуть знаходитись на відстані в 2 кроки, і навіть до зовсім не з’єднаних між собою. Цю концепцію відображає **мостовий проліт** (bridge span) — відстань між кінцевими точками ребра, якщо це ребро видалити. Ребра з великим прольотом з’єднують віддалені частини мережі, тому їх можна вважати слабкими зв’язками, незважаючи на те, що вони відіграють важливу роль.

Наступний код обчислює довжину кожного ребра в мережі карате-клубу:

```
def bridge_span(G):
# Отримуємо список ребер
    edges = G.edges()
# Створюємо копію графа
    G = nx.Graph(G)
# Створюємо словник для збереження результату
    result = dict()
    for v, w in edges:
# Тимчасово видаляємо ребро
        G.remove_edge(v, w)
# Знаходимо нову відстань між двома вузлами
# після видалення ребра
        try:
            d = nx.shortest_path_length(G, v, w)
            result[(v, w)] = d
        except nx.NetworkXNoPath:
            result[(v, w)] = float('inf')
# Відновлюємо ребро
        G.add_edge(v, w)
    return result

span = bridge_span(G)
```

13.1.5.8 Порівняння міцності та прольоту

Розглянемо 10 найміцніших і 10 найслабших ребер у мережі карате-клубу. Наступний код виводить ці ребра:

```
# Упорядковуємо ребра за силою зв'язності
ordered_edges = sorted(strength.items(), key=lambda x: x[1])
print('Ребро\t Міцність\t Проліт\t Внутрішній зв\'язок')
# Виводимо 10 найміцніших
for e, edge_strength in ordered_edges[:10]:
   print('{:10}{}\t\t{}\t{}'.format(
   str(e), edge_strength, span[e], G.edges[e]['internal']))
print('...')
```

```
# Виводимо 10 найслабших
for e, edge_strength in ordered_edges[-10:]:
    print('{:10}{}\t\t{}\t{}'.format(
    str(e), edge_strength, span[e], G.edges[e]['internal']))
```

```
Ребро    Міцність    Проліт  Внутрішній зв'язок
(0, 11)   1     inf True
(0, 31)   1     3   False
(1, 30)   1     3   False
(2, 9)    1     3   False
(2, 27)   1     3   False
(2, 28)   1     3   False
(9, 33)   1     3   True
(13, 33)  1     3   False
(19, 33)  1     3   False
(23, 25)  1     3   True
...
(8, 32)   4     2   True
(23, 33)  4     2   True
(29, 33)  4     2   True
(1, 2)    5     2   True
(1, 3)    5     2   True
(2, 3)    5     2   True
(0, 2)    6     2   True
(0, 3)    6     2   True
(0, 1)    8     2   True
(32, 33)  11    2   True
```

Результат показує, що ребра з низькою міцністю і великим прольотом, як правило, є зовнішніми, з'єднуючи членів клубу, які розкололися на різні клуби-відколи. З іншого боку, ребра з високою міцністю і малим прольотом є внутрішніми, вони з'єднують членів клубу, які залишилися разом після розколу.

13.1.5.9 Спектральні міри складності

Спектром графа G називається множина власних значень матриці, що відповідає даному графу. Відомі декілька підходів встановлення зв'язку між графом G та його спектром. Для випадку регулярних графів (якими є графи часових рядів фондових індексів) можна показати, що різні види спектрів еквівалентні, тобто містять однакову кількість інформації щодо структури графа G.

Ми вже згадували, що одним із способів представлення графа у вигляді матриці є матриця суміжності. **Матриця Лапласа** (Laplacian matrix) L — також є одним видів подання графа. Вона може бути використана для розрахунку кількості остовних дерев для графа. Для знаходження матриці Лапласа використовують формулу $L = D - A$, де D — діагональна матриця:

$$d_{ij} = \begin{cases} d_i, & i = j, \\ 0, & i \neq j, \end{cases}$$

де d_i — ступінь відповідної вершини графа. Отже,

$$l_{ij} = \begin{cases} d_i, & i = j, \\ -1, & i \neq j \text{ і } v_i \text{ суміжна з } v_j, \\ 0, & \text{в іншому випадку.} \end{cases}$$

Алгебраїчна зв'язність графа (algebraic connectivity) — друге найменше власне значення матриці Лапласа. Це власне значення більше нуля тоді і тільки тоді, коли граф зв'язний. Величина цього значення відображає, наскільки зв'язним є даний граф, і використовується при аналізі надійності та синхронізації мереж. Бібліотека `NetworkX` містить метод `algebraic_connectivity()` для обчислення даного показника. Бібліотека також надає змогу розрахувати нормалізовану матрицю Лапласа. Сенс нормалізації полягає в тому, що вершина з великим ступенем вершини, яку також називають **важкою вершиною**, призводить до того, що в матриці Лапласа з'являється великий діагональний елемент, який домінує у властивостях матриці. Нормалізація спрямована на те, щоб зробити вплив таких вершин більш рівним впливу інших вершин шляхом ділення елементів матриці Лапласа на ступені вершин. Щоб уникнути ділення на нуль, ізольовані вершини з нульовими ступенями виключаються з процесу нормалізації.

```
print("Алгебраїчна зв'язність для карате-клубу: ",
nx.algebraic_connectivity(G_karate, normalized=True, method='tracemin_lu'))
print("Алгебраїчна зв'язність для роману: ", nx.algebraic_connectivity(G_novel,
normalized=True, method='tracemin_lu'))
print("Алгебраїчна зв'язність для жінок: ", nx.algebraic_connectivity(G_woman,
normalized=True, method='tracemin_lu'))
```

```
Алгебраїчна зв'язність для карате-клубу:  0.11007419200657863
Алгебраїчна зв'язність для роману:  0.06737737553000264
Алгебраїчна зв'язність для жінок:  0.2079721479691762
```

Можемо бачити, що найбільш зв'язним у даному випадку представляється саме граф жінок. Тобто спілкування та кооперація між ними залишається найбільш тісною.

Енергія графа (graph energy) — це сума абсолютних значень власних значень матриці суміжності графа. Нехай G є граф з n вершинами. Передбачається, що G — простий, тобто він не містить петлі чи паралельних ребер. Нехай A — матриця суміжності графа G і λ_i, $i = 1, \ldots, n$ — власні значення матриці A. Тоді енергія графа визначається як

$$E(G) = \sum_{i=1}^{n} |\lambda_i|.$$

Вбудованого методу в NetworkX для визначення енергії графа немає, але ми доволі запросто можемо розрахувати спектр власних значень матриці суміжності, а потім скористатися формулою вище. Власні значення матриці A можна знайти за допомогою методу adjacency_spectrum(). Далі визначимо наступну функцію для розрахунку енергії графа:

```
def graph_energy(G):

    adj_spectrum = nx.adjacency_spectrum(G) # спектр власних значень матриці суміжності
    graph_en = np.sum(np.abs(adj_spectrum))

    return graph_en
```

Тепер розрахуємо енергію для кожного досліджуваного графа:

```
print("Енергія графа карате-клубу: ", graph_energy(G_karate))
print("Енергія графа роману: ", graph_energy(G_novel))
print("Енергія графа жінок: ", graph_energy(G_woman))
```

```
Енергія графа карате-клубу:  153.22817810462595
Енергія графа роману:  460.4651813130986
Енергія графа жінок:  51.82012198561654
```

Найвище значення енергії графа вказує на найвищу складність мережі або на найвищий ступінь централізованості деяких вузлів. Для наших графів видно, що найвища енергія приходить саме граф персонажів роману. Тобто, тут є декілька персонажів, на які приходить найбільша кількість зв'язків (діалогів) у порівнянні з іншими персонажами.

Спектральний розрив (spectral gap) — різниця між найбільшим і другим за величиною власними значеннями, надає інформацію про те, як швидко досягається синхронний стан. Можемо визначити й прорахувати наступну функцію:

```
def spectral_gap(G):
    adj_spectrum = nx.adjacency_spectrum(G)
    sorted_adj_spectrum = np.sort(adj_spectrum.real)
    spec_gap = sorted_adj_spectrum[-1] - sorted_adj_spectrum[-2]

    return spec_gap

print("Спектральний розрив для карате-клубу: ", spectral_gap(G_karate))
print("Спектральний розрив для роману: ", spectral_gap(G_novel))
print("Спектральний розрив для жінок: ", spectral_gap(G_woman))
```

```
Спектральний розрив для карате-клубу:  4.58124582340616
Спектральний розрив для роману:  16.258106786753928
Спектральний розрив для жінок:  2.3618098280048976
```

Спектральний радіус (spectral radius) є найбільшим за модулем власним значенням:

$$r(A) = \max_{\lambda \in Spec(A)} |\lambda|,$$

де $Spec(A)$ — спектр власних значень матриці суміжності. Для розрахунків визначимо наступну функцію:

```
def spectral_radius(G):
    adj_spectrum = nx.adjacency_spectrum(G).real
    spec_rad = np.max(np.abs(adj_spectrum))

    return spec_rad

print("Спектральний радіус для карате-клубу: ", spectral_radius(G_karate))
print("Спектральний радіус для роману: ", spectral_radius(G_novel))
print("Спектральний радіус для жінок: ", spectral_radius(G_woman))
```

```
Спектральний радіус для карате-клубу:  21.687565903954198
Спектральний радіус для роману:  65.0262803552607
Спектральний радіус для жінок:  6.741908124910325
```

Спектральний момент (spectral moment). Для визначення k-ого спектрального моменту використовують матрицю суміжності. Визначимо її наступним чином:

$$m_k(A) = \frac{1}{n}\sum_{i=1}^{n} \lambda_i^k,$$

де λ_i — власні значення матриці суміжності A, n — кількість вершин графа G. Значення k у нашому випадку випадку буде дорівнювати 3. Тобто, будемо обчислювати спектральний момент 3-го порядку. Визначимо наступну функцію для розрахунку даного показника:

```
def spectral_moment(G):

    adj_spectrum = nx.adjacency_spectrum(G).real
    spec_mom_3 = np.mean(adj_spectrum**3)

    return spec_mom_3

print("Спектральний момент для карате-клубу: ", spectral_moment(G_karate))
print("Спектральний момент для роману: ", spectral_moment(G_novel))
print("Спектральний момент для жінок: ", spectral_moment(G_woman))
```

```
Спектральний момент для карате-клубу:  321.3529411764713
Спектральний момент для роману:  4325.688311688335
Спектральний момент для жінок:  1.1585177261963508e-13
```

Останні показники говорять по те, що персонажі роману характеризуються найвищим ступенем складності в порівнянні з іншими графами. Ми показали, що достатня кількість вузлів має досить невисокий найкоротший шлях, але може мати гіршу щільність зв'язності вузлів або рівнорозподіленності ступеня впливовості.

13.1.5.10 Проблема малого світу

У 1967 році соціальні психологи Джеффрі Треверс і Стенлі Мілґрем надіслали листи групам людей у Вічіті, штат Канзас, та Омасі, штат Небраска. Вони також обрали одну цільову особу в штаті Массачусетс. Кожному отримувачу листа було доручено переслати його знайомому, який, найімовірніше, знав цільову людину. Багато листів дійшли до адресата, і дослідники змогли з'ясувати, скільки кроків було зроблено для цього. Середня кількість кроків становила шість, звідси і поширена фраза "шість ступенів відокремлення" [213].

13.1.5.11 Кільцеві мережі

Як правило, більшість знайомих людини — це люди, які живуть у тій самій місцевості. Якби кожна людина була знайома лише з тими, хто живе поруч, то можна було б очікувати, що для того, щоб надіслати повідомлення з Канзасу до Массачусетсу, знадобилося б більше шести кроків, оскільки кожен крок міг би подолати лише невелику відстань. Таку мережу можна змоделювати як кільце: вузли, розташовані по колу, причому кожен вузол з'єднаний з найближчими $k/2$ вузлами з кожного боку. Наступний приклад створює та візуалізує чотирикільце за допомогою функції `watts_strogatz_graph()` про яку ми ще поговоримо.

```
G_small_ring = nx.watts_strogatz_graph(16, 4, 0)
pos = nx.circular_layout(G_small_ring)

plt.figure(figsize=(6, 4))
nx.draw_networkx(G_small_ring, pos=pos, with_labels=False)
```

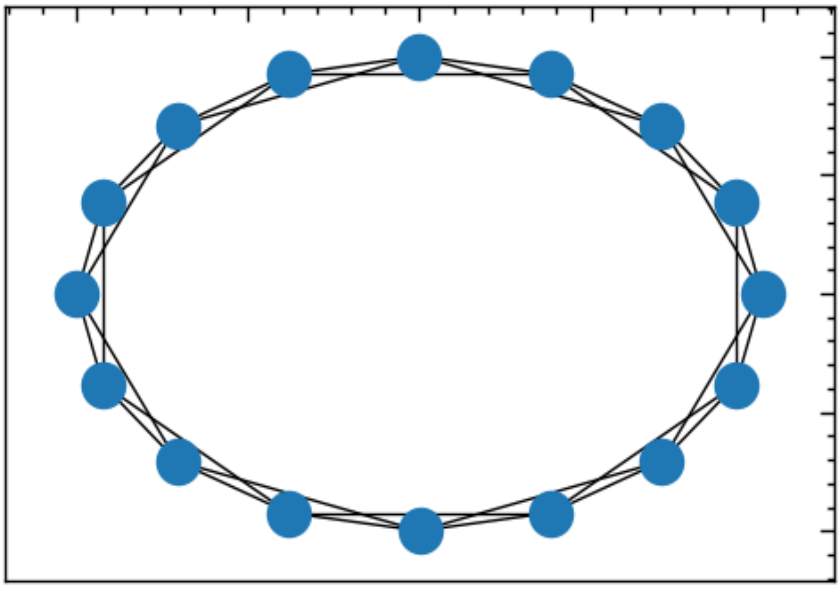


Рис. 13.36: Кільцеве представлення графа Воттса-Строгаца

Щоб з'єднати два вузли в попередньому прикладі, потрібно пройти по краю кола, пропускаючи щонайбільше кожен другий вузол. Навіть у цій дуже маленькій мережі типова мережна відстань є досить великою.

Наступний код знаходить середній найкоротший шлях і середню кластеризацію в більш реалістичному 10-ти кільцевому графі з 4000 вузлів:

```
G_ring = nx.watts_strogatz_graph(4000, 10, 0)
nx.average_shortest_path_length(G_ring)
```

```
200.45011252813202
```

```
nx.average_clustering(G_ring)
```

```
0.6666666666666546
```

Ця мережа має в середньому 200 кроків розділення, що набагато більше, ніж шість! Вона також має досить великий середній коефіцієнт кластеризації 0.67, що показує, що сусіди вузла мають тенденцію бути пов'язаними один з одним.

13.1.5.12 Випадкові мережі

Щоб дослідити цю таємницю, розглянемо інший тип мережі. У цій мережі ми починаємо з k-кільця, але випадковим чином переставляємо кінцеві точки кожного ребра. В результаті отримаємо мережу з тією ж кількістю вузлів і ребер, але з випадковою структурою, що демонструється наступним кодом:

```
G_small_random = nx.watts_strogatz_graph(16, 4, 1)
pos = nx.circular_layout(G_small_random)
```

```
plt.figure(figsize=(6, 4))
nx.draw_networkx(G_small_random, pos=pos, with_labels=False)
```

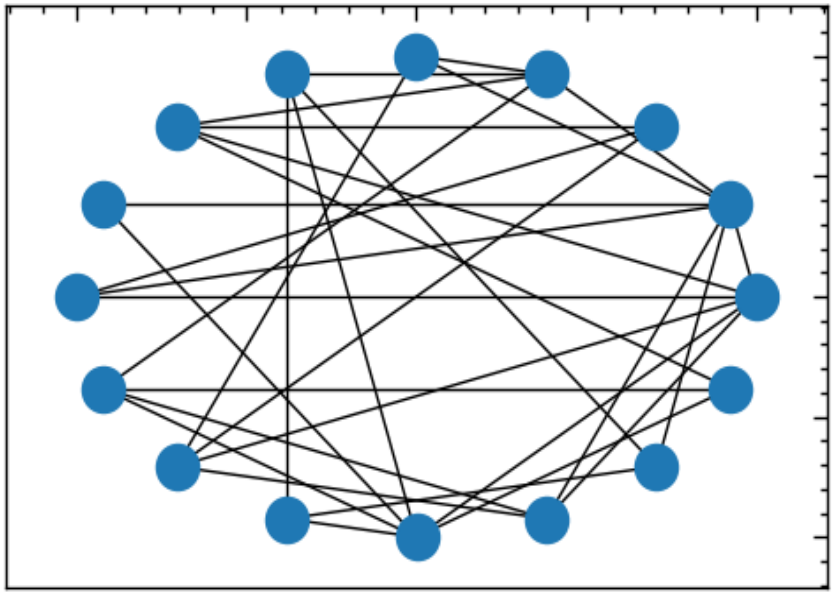


Рис. 13.37: Кільцеве представлення графа Воттса-Строгаца з випадковим перев'язуванням ребер

Тепер давайте розглянемо властивості перев'язаного 10-ти кільцевого графа з 4000 вузлів:

```
G_random = nx.watts_strogatz_graph(4000, 10, 1)
nx.average_shortest_path_length(G_random)
```

```
3.867028507126782
```

```
nx.average_clustering(G_random)
```

```
0.0020522294183284917
```

Середній найкоротший шлях дуже близький до реальної соціальної мережі, але середня кластеризація тепер майже 0. Поки що моделі, які ми бачили, досягають коротких шляхів або високої кластеризації, але не того й іншого разом.

13.1.5.13 Мережа Воттса-Строгаца

Проблема малого світу полягає в тому, як люди, що живуть на великій відстані один від одного, можуть бути пов'язані короткими шляхами, навіть якщо їхні зв'язки є локальними. Дункан Воттс і Стівен Строгац розробили клас мереж для пояснення такої поведінки. Мережі починаються як k-кільця: вузли, розміщені по колу, кожен з яких з'єднаний з найближчими k сусідами. Потім, з ймовірністю p, ребра кожного вузла перев'язуються з іншим випадково обраним вузлом. Ці перестановки створюють короткі шляхи по всій мережі. Навіть

невелика кількість коротких шляхів значно скорочує відстані між вузлами мережі, вирішуючи проблему малого світу. Фактично, це саме те, що робить функція `watts_strogatz_graph()`, яку ми використовували, а третій параметр задає частку ребер, які потрібно перезв'язати. Наступний код обчислює середній найкоротший шлях і середню кластеризацію для діапазону ймовірностей перезв'язування:

```
path = []
clustering = []
# Пробуємо список імовірностей перезв'язування
p = [10**(x) for x in range(-6, 1)]
for p_i in p:
    path_i = []
    clustering_i =[]
# Створюємо 10 моделей для кожної ймовірності
    for n in range(10):
        G = nx.watts_strogatz_graph(1000, 10, p_i)
        path_i.append(nx.average_shortest_path_length(G))
        clustering_i.append(nx.average_clustering(G))
# Усереднюємо показники для кожного значення p_i
    path.append(sum(path_i) /len(path_i))
    clustering.append(sum(clustering_i) /len(clustering_i))
```

Результати наступного коду зберігаються у списках `path` та `clustering`. Використовуючи функцію `semilogx()` з `matplotlib.pyplot`, наступний код візуалізує, як ці значення змінюються при зміні ймовірності перев'язування від 0 до 1:

```
plt.figure(figsize=(6, 4))

plt.semilogx(p, [x / path[0] for x in path], label=r'$L_{mean} / L_0$')
plt.semilogx(p, [x / clustering[0] for x in clustering], label=r'$C_{mean} /
C_0$')
plt.tick_params(axis='both', which='major', labelsize=16)
plt.xlabel(r"Імовірність перев'язування $p$", fontsize=16)
plt.legend();
```

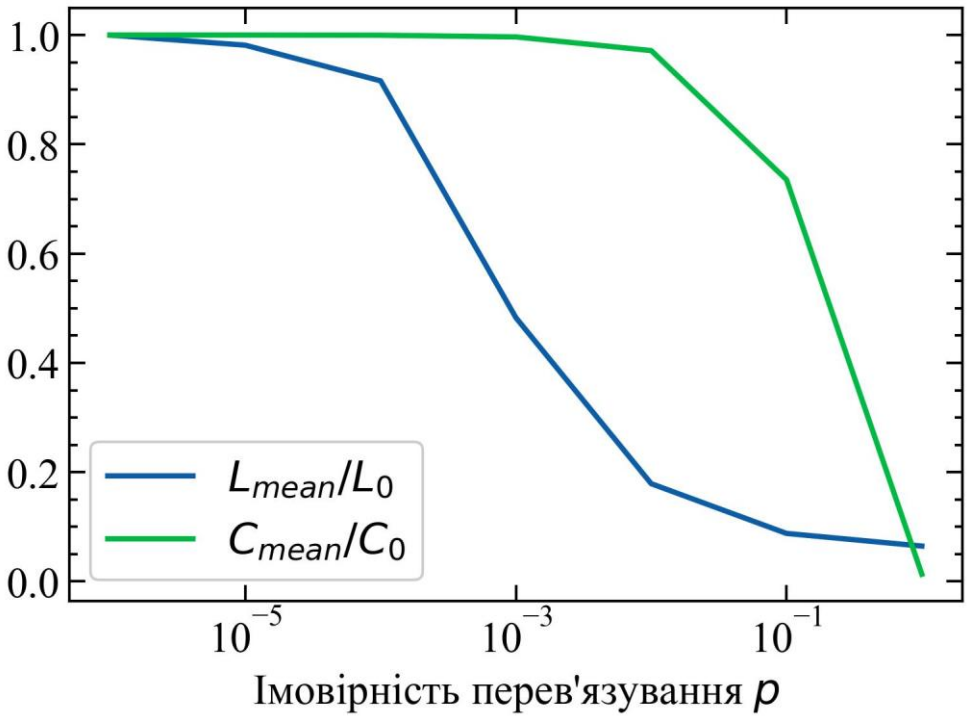


Рис. 13.38: Зміна відносного середнього найкоротшого шляху та коефіцієнту кластеризації від імовірності перев’язування графа

Як ми вже бачили, зі збільшенням кількості перезв’язувань, як середня кластеризація, так і середній найкоротший шлях зменшуються. Однак цікава річ відбувається при проміжних значеннях. Довжина шляху стає коротшою при дуже низьких значеннях перезв’язування, в той час як зменшення кластеризації відбувається лише при більших значеннях перезв’язування. Іншими словами, перезв’язування дуже малої частки ребер створює “мости”, які з’єднують віддалені частини мережі і різко скорочують середній найкоротший шлях, не змінюючи при цьому кластеризацію. Можна сказати, що найкращий тип мереж це той, що зберігає як частку впорядкованості, так і частку випадковості.

Далі можемо подивитись, як виглядає мережа Воттса й Строгаца при наступних імовірностях: $p = 0$, $p = 0.1$ та $p = 1$.

```
plt.figure(figsize=(8, 4))
for i, p in enumerate([0.0, 0.1, 1.0]):
# Генеруємо граф
    G = nx.watts_strogatz_graph(12, 6, p)
# Будуємо рисунок
    plt.subplot(1, 3, i +1)
    pos = nx.circular_layout(G)
    nx.draw_networkx(G, pos=pos)
    plt.title("p = {:0.1f}".format(p))
```

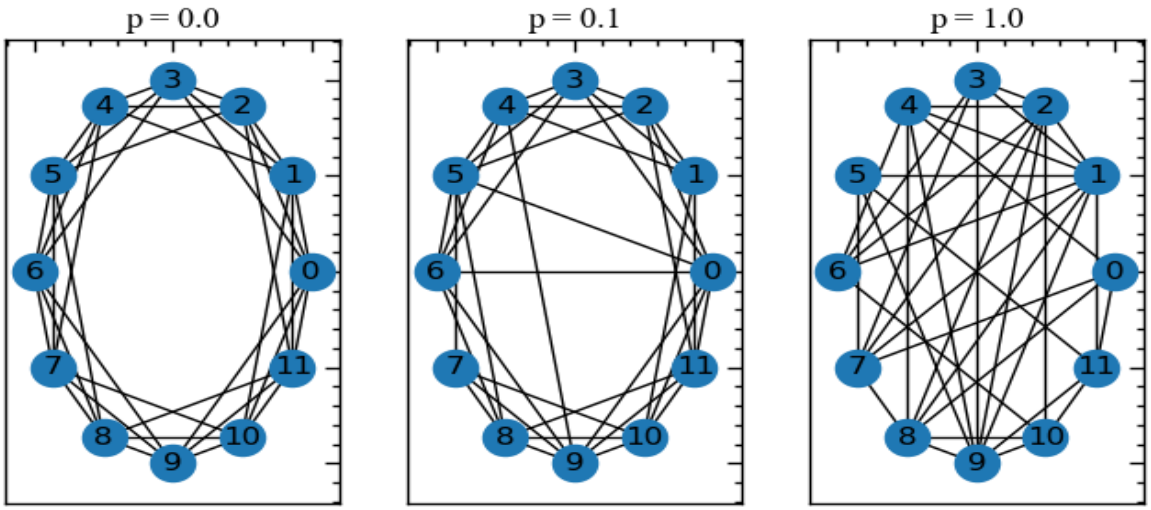


Рис. 13.39: Представлення графа Воттса й Строгаца при різних імовірностях перев'язування ребер

У деяких випадках перезв'язування може призвести до того, що дві компоненти в мережі Воттса-Строгаца будуть роз'єднані. Роз'єднана мережа може бути непотрібним ускладненням. Мережа Ньюмана-Воттса-Строгаца — це варіант, який гарантує, що отримана мережа буде зв'язною. Вона схожа на оригінальну версію, але залишає копію оригінального ребра на місці кожного ребра, що перезв'язується. Такі мережі можна створювати за допомогою функції `newman_watts_strogatz_graph()`, як показано нижче:

```
plt.figure(figsize=(8, 4))
for i, p in enumerate([0.0, 0.1, 1.0]):

    G = nx.newman_watts_strogatz_graph(12, 6, p)

    plt.subplot(1, 3, i +1)
    pos = nx.circular_layout(G)
    nx.draw_networkx(G, pos=pos)
    plt.title("p = {:0.1f}".format(p))
```

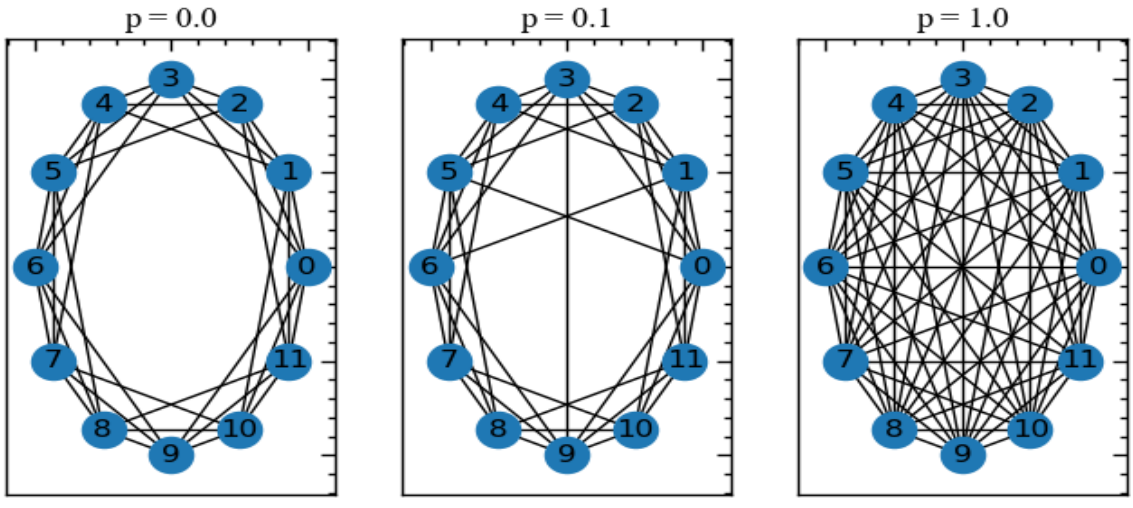


Рис. 13.40: Представлення графа Ньюмана-Воттса-Строгаца при різних імовірностях перезв'язування ребер

13.1.5.14 Степеневі закони та переважне приєднання

Від інтернету до поїздок в аеропорт, багато мереж характеризуються кількома вузлами з великою кількістю зв'язків і багатьма вузлами з дуже малою кількістю зв'язків. Такі мережі характеризуються **важкими хвостами**, тому що при побудові гістограми ступенів вузлів, вузли з високим рівнем зв'язності утворюють хвіст.

Існує багато способів генерування мереж з важким хвостом, але одним з найпоширеніших є модель **переважного приєднання** Барабаші-Альберт. Модель переважного приєднання імітує процеси, в яких багаті стають багатшими. Кожного разу, коли додається новий вузол, він випадковим чином з'єднується з існуючими вузлами, причому більш вірогідним є з'єднання з вузлами високого ступеня.

У NetworkX функція `barabasi_albert_graph()`, яка генерує мережі переважного приєднання. У наступному коді показано приклад такої мережі з 35 вузлами:

```
G_preferential_35 = nx.barabasi_albert_graph(35, 1)
pos = nx.spring_layout(G_preferential_35, k=0.1)

plt.figure(figsize=(6, 4))
nx.draw_networkx(G_preferential_35, pos)
```

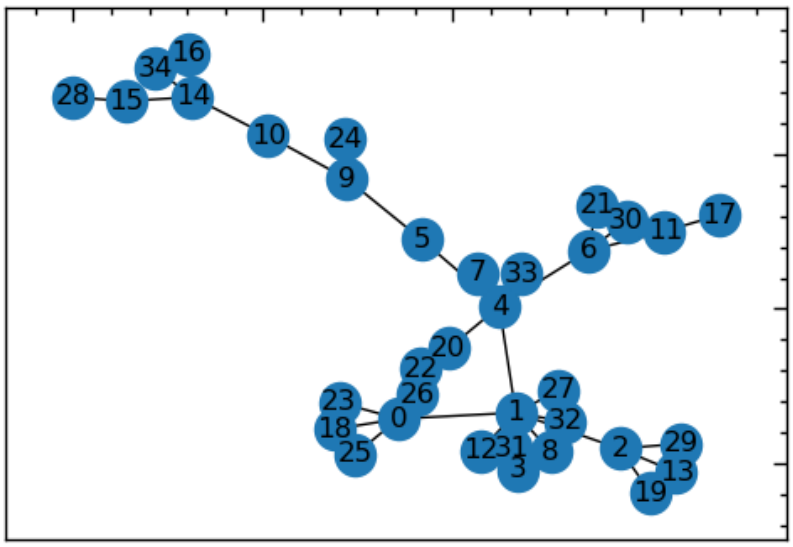


Рис. 13.41: Представлення графа переважного приєднання Барабаші-Альберт при 35 вузлах

Структура мережі переважного приєднання ще більш очевидна при більшій кількості вузлів. У наступному прикладі використовується 1000 вузлів:

```
G_preferential_1000 = nx.barabasi_albert_graph(1000, 1)
pos = nx.spring_layout(G_preferential_1000)

plt.figure(figsize=(6, 4))
nx.draw_networkx(G_preferential_1000, pos, node_size=0, with_labels=False)
```

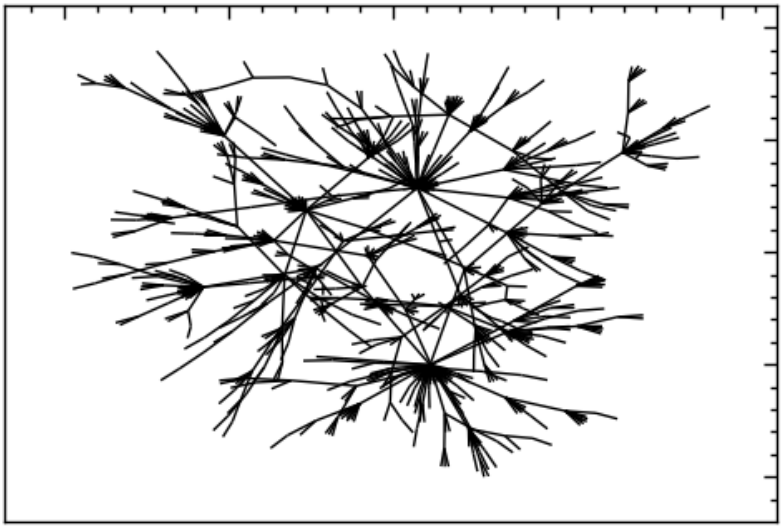


Рис. 13.42: Представлення графа переважного приєднання Барабаші-Альберт при 1000 вузлах

Важкі хвости цих мереж можна побачити, побудувавши їхні ступеневі розподіли. Наступна функція будує розподіл ступенів мережі:

```
def plot_degree_hist(G, title):
"""Функція для побудови розподілу ступенів вершин мережі"""
    plt.hist(dict(nx.degree(G)).values(), bins=range(1, 11))
    plt.xlabel('Ступінь')
    plt.ylabel('Щільність')
    plt.title(title)
```

Використовуючи цю функцію, наступний код візуалізує розподіл ступенів для 35-вузлової та 1000-вузлових мереж переважного приєднання:

```
plt.figure(figsize=(8, 5))
ax = plt.subplot(1, 2, 1)
plot_degree_hist(G_preferential_35, '35 вузлів')
for spine in ax.spines.values():
    spine.set_visible(True)
ax = plt.subplot(1, 2, 2)
for spine in ax.spines.values():
    spine.set_visible(True)
plot_degree_hist(G_preferential_1000, '1000 вузлів')
plt.tight_layout()
```

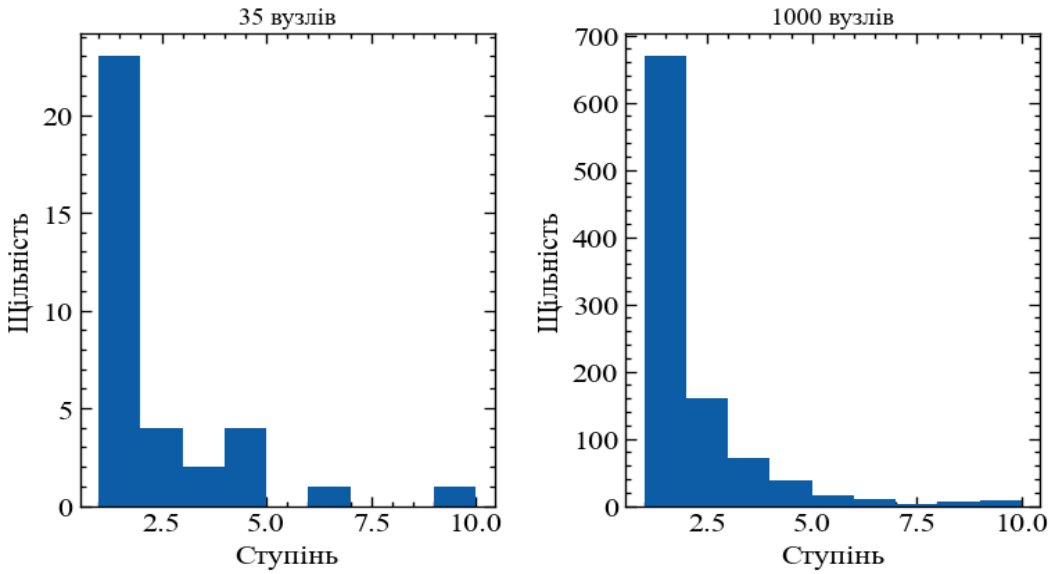


Рис. 13.43: Гістограми ступенів вершини графів переважного приєднання при 35 та 1000 вершинах

Мережі з переважним приєднанням мають одну цікаву властивість: вони **масштабно-інваріантні**. Розподіл ступенів у масштабоінваріантних мережах підпорядковується степеневому закону, що призводить до схожої структури на різних масштабах. Один із способів побачити це — порівняти попередні гістограми. Незважаючи на дуже різні масштаби, вони мають схожу форму. Розподіл ступенів вершин можна описати степеневою функцією виду:

$$P(k) \propto k^{-\gamma},$$

де k — ступінь вузла, $P(k)$ — ймовірність того, що вузол має ступінь k, і γ — показник степеневого закону. Показник γ зазвичай знаходиться в діапазоні від 2 до 3 для більшості реальних мереж.

Розподіл ступенів степеневого закону має важливі наслідки для структури та функцій мереж. Наприклад, мережі зі степеневим розподілом часто є більш надійними і стійкими до випадкових збоїв, але більш вразливими до цілеспрямованих атак на вузли з високим ступенем.

13.2 Хід роботи

Тепер давайте проведемо порівняльний аналіз графів різної складності з використанням деяких із зазначених показників. За допомогою бібліотеки `NetworkX` розглянемо наступні типи графів:

- лінійний граф — `path_graph()`;
- циклічний граф — `cycle_graph()`;

- граф-зірка — `star_graph()`;
- граф Ердеша-Реньї — `erdos_renyi_graph()`;
- граф малого світу — `watts_strogatz_graph()`;
- граф переважного приєднання — `barabasi_albert_graph()`.

Візуалізуємо кожен із зазначених графів:

```
fig, axes = plt.subplots(3, 2, figsize=(10, 8))

# лінія
axes[0, 0].set_title('Лінія')
line_graph = nx.path_graph(100)
pos_line_graph = nx.spring_layout(line_graph, k=0.15, iterations=100)
nx.draw_networkx(line_graph, pos=pos_line_graph, node_size=10,
with_labels=False, ax=axes[0, 0])

# коло
axes[0, 1].set_title('Коло')
cycle_graph = nx.cycle_graph(100)
pos_cycle_graph = nx.circular_layout(cycle_graph)
nx.draw_networkx(cycle_graph, pos=pos_cycle_graph, node_size=10,
with_labels=False, ax=axes[0, 1])

# зірка
axes[1, 0].set_title('Зірка')
star_graph = nx.star_graph(100)
pos_star_graph = nx.spring_layout(star_graph, k=0.15, iterations=100)
nx.draw_networkx(star_graph, pos=pos_star_graph, node_size=10,
with_labels=False, ax=axes[1, 0])

# Ердеша-Реньї
axes[1, 1].set_title('Ердеш-Реньї')
erdos_renyi_graph = nx.erdos_renyi_graph(100, 0.01)
pos_erdos_renyi_graph = nx.circular_layout(erdos_renyi_graph)
nx.draw_networkx(erdos_renyi_graph, pos=pos_erdos_renyi_graph, node_size=10,
with_labels=False, ax=axes[1, 1])

# Малий світ
axes[2, 0].set_title('Малий світ')
small_world_graph = nx.watts_strogatz_graph(100, 30, 0.01, seed=32)
pos_small_world_graph = nx.spring_layout(small_world_graph, k=0.15,
iterations=100)
nx.draw_networkx(small_world_graph, pos=pos_small_world_graph, node_size=10,
with_labels=False, ax=axes[2, 0])

# Переважне приєднання
axes[2, 1].set_title('Переважне приєднання')
barabasi_albert_graph = nx.barabasi_albert_graph(100, 30, seed=32)
pos_barabasi_albert_graph = nx.spring_layout(barabasi_albert_graph, k=0.15,
iterations=100)
nx.draw_networkx(barabasi_albert_graph, pos=pos_barabasi_albert_graph,
node_size=10, with_labels=False, ax=axes[2, 1])
```

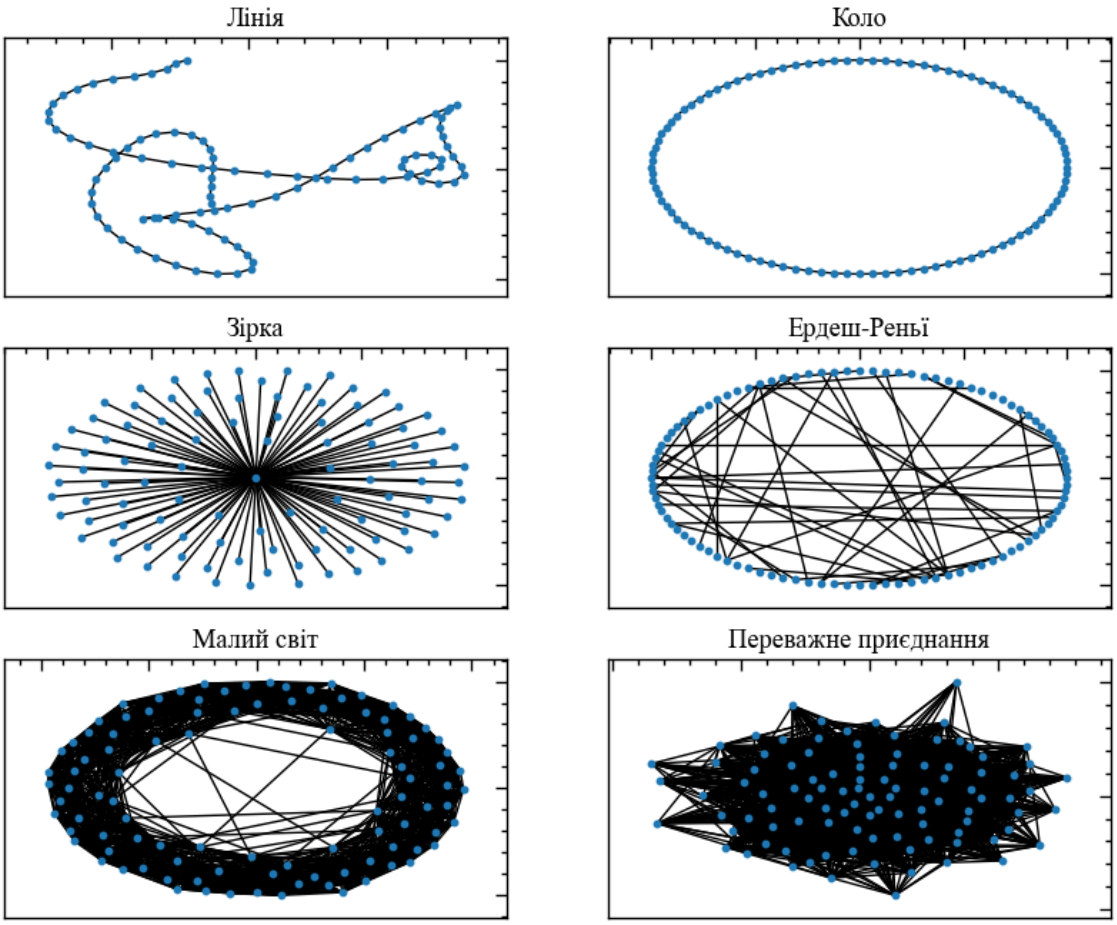


Рис. 13.44: Візуалізація графів: лінія, коло, зірка, Ердеш-Реньї, Малий світ і Переважне приєднання

Кожен із даних графів може різнитись за своїми спектральними і топологічними властивостями: деякі можуть мати вищий ступінь кластеризації, ступеня вершини, посередництва тощо. Розглянемо як ранжується ступінь складності кожного графа за досліджуваними нами показниками.

Спочатку збережемо кожен із побудованих графів до одного масиву для ітеративного проведення розрахунків по кожному з них:

```
graphs = [line_graph, cycle_graph, star_graph, erdos_renyi_graph,
small_world_graph, barabasi_albert_graph] # графи
labels = ['Лінія', 'Коло', 'Зірка', 'Ердеш-Реньї', 'Малий світ', 'Переважне
приєднання']                    # їх мітки
colors = ['b', 'purple', 'red', 'green', 'pink', 'black']
linestyles = ['-', '-', '--', '--', ':', '-']
markers = ['d', 'v', '*', 's', 'H', 'o']
```

13.2.1 Спектральні міри складності

Тепер виконаємо розрахунки спектральних мір складності для кожного графа:

```
algebraic_connect_vals = np.zeros(6)
energy_vals = np.zeros(6)
spec_gap_vals = np.zeros(6)
spec_mom_vals = np.zeros(6)

for i, graph in enumerate(graphs):
    algebraic_connect_vals[i] = nx.algebraic_connectivity(graph,
normalized=False, method='tracemin_lu')
    energy_vals[i] = graph_energy(graph)
    spec_gap_vals[i] = spectral_gap(graph)
    spec_mom_vals[i] = spectral_moment(graph)

num_rep =30
algebraic_connect_vals = np.repeat(algebraic_connect_vals[:, np.newaxis],
num_rep, axis=1)
energy_vals = np.repeat(energy_vals[:, np.newaxis], num_rep, axis=1)
spec_gap_vals = np.repeat(spec_gap_vals[:, np.newaxis], num_rep, axis=1)
spec_mom_vals = np.repeat(spec_mom_vals[:, np.newaxis], num_rep, axis=1)
```

Виведемо результат:

```
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Зв'язність
axes[0, 0].set_title('Алгебраїчна зв\'язність')
for i in range(len(graphs)):
    axes[0, 0].plot(algebraic_connect_vals[i],
                    color=colors[i],
                    marker=markers[i],
                    linestyle=linestyles[i],
                    label=labels[i])
axes[0, 0].legend(fontsize=13)


# Енергія
axes[0, 1].set_title('Енергія графа')
for i in range(len(graphs)):
    axes[0, 1].plot(energy_vals[i],
                    color=colors[i],
                    marker=markers[i],
                    linestyle=linestyles[i],
                    label=labels[i])
axes[0, 1].legend(fontsize=13)

# Розрив
axes[1, 0].set_title('Спектральний розрив')
for i in range(len(graphs)):
    axes[1, 0].plot(spec_gap_vals[i],
                    color=colors[i],
                    marker=markers[i],
                    linestyle=linestyles[i],
                    label=labels[i])
axes[1, 0].legend(fontsize=13)

# Момент
```

```
axes[1, 1].set_title('Спектральний момент')
for i in range(len(graphs)):
    axes[1, 1].plot(spec_mom_vals[i],
                    color=colors[i],
                    marker=markers[i],
                    linestyle=linestyles[i],
                    label=labels[i])
axes[1, 1].legend(fontsize=13)

plt.show();
```

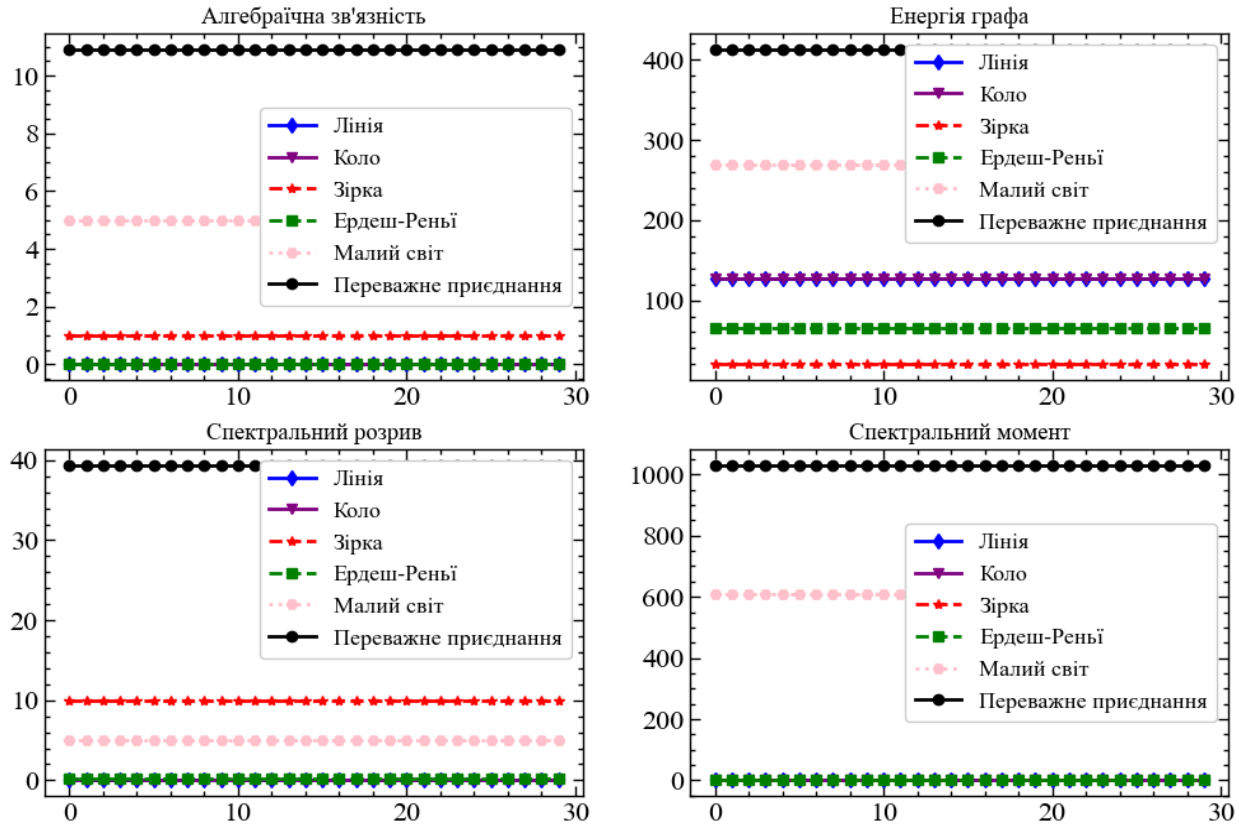


Рис. 13.45: Спектральні властивості канонічних і модельних графів: алгебраїчної зв’язності, енергії графа, спектрального розриву та спектрального моменту

З рисунку (Рис. 13.45) можна побачити наступне:

- по-перше, усі спектральні показники залишаються найбільшими саме для графа переважного приєднання, що представляється найбільш складним серед усіх інших графів;
- по друге, згідно динаміки спектральних показників, найпростішими серед усіх графів є граф лінії, зірки та Ердеша-Реньї. Для лінії зберігається зв’язок тільки між парами послідовних вершин. Для зірки зберігається зв’язок усіх вершин із центром, але самі вони не пов’язані один із одним;
- по третє, граф малого світу залишається другим за складністю майже за всіма показниками, окрім спектрального розриву. Спектральний розрив

говорить, що граф зірки є трохи складнішим за граф малого світу. Це може бути обумовлене тим, що для зірки ми спостерігаємо достатньо високий ступінь централізації.

13.2.2 Топологічні міри

Розрахуємо для досліджуваних графів топологічні міри складності. В якості прикладу розглянемо такі міри як

- максимальний ступінь вершини (d_{max});
- глобальний коефіцієнт кластеризації (C);
- середній ступінь посередництва (B_{mean});
- середня довжина найкоротшого шляху (L_{mean}).

```
max_degree_vals = np.zeros(6)
global_clust_vals = np.zeros(6)
mean_betweenness_vals = np.zeros(6)
mean_path_vals = np.zeros(6)

for i, graph in enumerate(graphs):
    max_degree_vals[i] = max(dict(graph.degree()).values())
    global_clust_vals[i] = nx.average_clustering(graph)
    mean_betweenness_vals[i] =
np.mean(list(nx.betweenness_centrality(graph).values()))
    mean_path_vals[i] = np.mean([nx.average_shortest_path_length(C) for C in
                                 (graph.subgraph(c).copy() for c in
nx.connected_components(graph))])

num_rep = 30
max_degree_vals = np.repeat(max_degree_vals[:, np.newaxis], num_rep, axis=1)
global_clust_vals = np.repeat(global_clust_vals[:, np.newaxis], num_rep, axis=1)
mean_betweenness_vals = np.repeat(mean_betweenness_vals[:, np.newaxis], num_rep,
axis=1)
mean_path_vals = np.repeat(mean_path_vals[:, np.newaxis], num_rep, axis=1)
```

Виводимо результат:

```
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

axes[0, 0].set_title('Макс. ступінь вершини')

for i in range(len(graphs)):
    axes[0, 0].plot(max_degree_vals[i],
                    color=colors[i],
                    marker=markers[i],
                    linestyle=linestyles[i],
                    label=labels[i])
axes[0, 0].legend(fontsize=13)

# Енергія
axes[0, 1].set_title('Глобальний коефіцієнт кластеризації')
for i in range(len(graphs)):
```

```
    axes[0, 1].plot(global_clust_vals[i],
                    color=colors[i],
                    marker=markers[i],
                    linestyle=linestyles[i],
                    label=labels[i])
axes[0, 1].legend(fontsize=13)

# Розрив
axes[1, 0].set_title('Середній ступінь посередництва')
for i in range(len(graphs)):
    axes[1, 0].plot(mean_betweenness_vals[i],
                    color=colors[i],
                    marker=markers[i],
                    linestyle=linestyles[i],
                    label=labels[i])
axes[1, 0].legend(fontsize=13)

# Момент
axes[1, 1].set_title('Середня довжина найкоротшого шляху')
for i in range(len(graphs)):
    axes[1, 1].plot(mean_path_vals[i],
                    color=colors[i],
                    marker=markers[i],
                    linestyle=linestyles[i],
                    label=labels[i])
axes[1, 1].legend(fontsize=13)

plt.show();
```

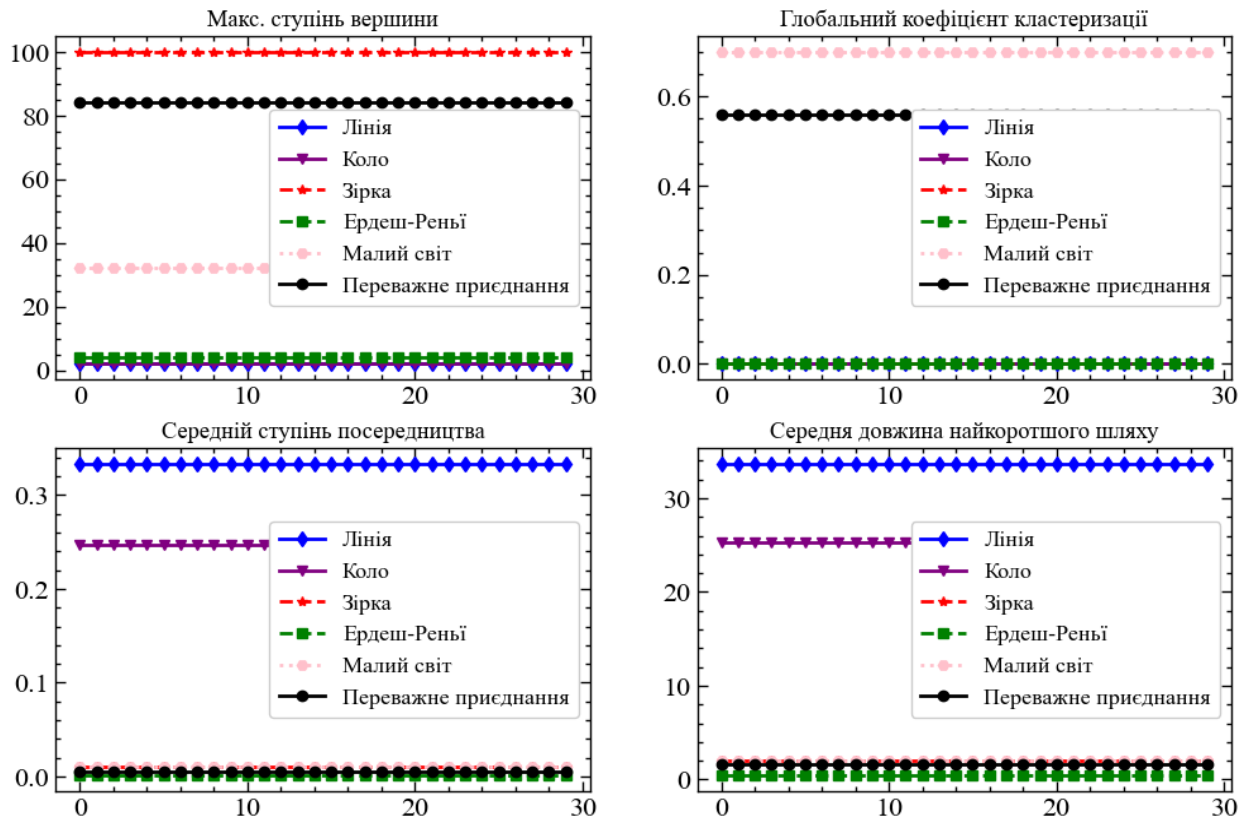


Рис. 13.46: Топологічні міри складності канонічних і модельних графів:

максимального значення ступеня вершини, глобального коефіцієнту кластеризації, середнього ступеня посередництва та середньої довжини найкоротшого шляху

На рисунку (Рис. 13.46) можна побачити наступне:

- по-перше, найбільшим максимальним ступенем вершини характеризується саме граф-зірка, центр якої з'єднаний абсолютно з усіма вершинами мережі. Другим по ступенем концентрованності йде граф переважного приєднання, що, як ми вже зазначали, є найкращою моделлю реальних соціальних систем. До найпростіших можна віднести графи лінії, кола та Ердеша-Реньї;
- по-друге, глобальний коефіцієнт кластеризації вказує на те, що найвищий ступінь кластеризації спостерігається саме для графа малого світу. Закономірно за ним іде граф переважного приєднання. Найпростішими знову виявляються графи Ердеша-Реньї, лінії, кола та, цього разу, зірки. Для зірки навіть візуально видно, що всі вершини мають тенденцію слідувати тільки за однією конкретною;
- по-третє, середній ступінь посередництва є найнижчим для зірки, графа Ердеша-Реньї, малого світу та переважного приєднання. Для цих мереж передача інформації від одного вузла до іншого не займає значну частку часу. Для лінії та кола від одного кінця графа до іншого може знадобитися досить великий проміжок часу для передачі інформації. Схожа ситуація спостерігається й для середньої довжини найкоротшого шляху, оскільки міра посередництва на пряму залежить від значення найкоротшого шляху від одного вузла до іншого.

13.3 Висновок

У даній лабораторній роботі було здійснено вступ до теорії графів і різних кількісних показників, що вона надає. На прикладі простих графів і мереж реального світу було показано, що графові показники дозволяють кількісно визначити ступінь ефективності, концентрованості, кластеризації, зв'язності тощо. Було показано, що графам реального світу властиві степенева залежність розподілу ступенів вершин, низька довжина найкоротшого шляху та високий ступінь кластеризації. У лабораторній 14 буде розширено спектр мережних показників і продемонстровано змогу їх використання в якості індикаторів-передвісників крахових подій.

13.4 Завдання для самостійної роботи

1. Проаналізуйте аналогічно інші з розрахованих мір складності як спектральних, так і топологічних
2. Вкажіть і аргументуйте, які з них, на вашу думку, кількісно описують складність досліджуваних мереж?
3. Побудуйте залежність різних мережних показників для часового ряду реального світу по аналогії з двома попередніми рисунками

14. Лабораторна робота № 14

Тема. Графодинамічний аналіз часових рядів

Мета. Навчитися використовувати елементи теорії графів для отримання спектральних і топологічних мір складності систем, що характеризуються часовими рядами

14.1 Теоретичні відомості

У попередній роботі ми ввели поняття мір складності для найпростіших графів та поширених мережних моделей, порівняли деякі з мір складності. У даній роботі ми продемонструємо сучасні методи перетворення часових рядів у мережу (граф) з подальшим дослідженням відповідних спектральних і топологічних мір складності. Ми також покажемо, що вказані міри можна співставляти з динамікою вихідного часового ряду (звідси **графодинаміка**) і якщо вони є інформативними щодо можливих змін власне ряду, то їх можливо використовувати для побудови індикаторів характерної динаміки складних систем.

Більшість складних систем інформують про свою структурну та динамічну природу, генеруючи послідовність певних характеристик, що можна представити часовими рядами. Останніми роками розроблено цікаві алгоритми перетворення часових рядів у мережу, що дозволяє розширити діапазон відомих характеристик часових рядів навіть до мережних [214–216]. Останнім часом було запропоновано декілька підходів до перетворення часових послідовностей у складні мережеподібні відображення. Ці методи можна умовно розділити на три класи [217]. Перший базується на вивченні “видимості” послідовних значень часового ряду і називається **графом видимості** (Visibility Graph, VG) [217,218].

Другий аналізує взаємне наближення різних відрізків часової послідовності і використовує техніку рекурентного аналізу [217] (див. лабораторні 2 і 3). Рекурентна діаграма відображає існуючу повторюваність фазових траєкторій у вигляді бінарної матриці, елементами якої є одиниці або нулі, залежно від того, чи є близькими (рекурентними) із заданою точністю чи ні обрані точки фазового простору динамічної системи. Рекурентна діаграма легко трансформується в матрицю суміжності, за якою розраховуються спектральні та топологічні характеристики графа [219].

Нарешті, якщо в основу формування зв'язків елементів графа покласти кореляційні відношення між ними, то отримаємо кореляційний граф [217]. Для побудови та аналізу властивостей кореляційного графа необхідно сформувати з кореляційної матриці матрицю суміжності. Для цього необхідно ввести величину, яка для кореляційного поля буде слугувати відстанню між корельованими агентами. Така відстань може бути представлена як $d_{ij} = \sqrt{2(1 - C_{ij}}$, де C_{ij} — це коефіцієнт кореляції між двома активами. Так, якщо коефіцієнт кореляції між двома активами значний, то відстань між ними невелика, і, починаючи з певного критичного значення d_{cr}, активи можна вважати зв'язаними на графі. Для матриці суміжності це означає, що вони є суміжними на графі. В іншому випадку активи не є суміжними. У цьому випадку умова зв'язності графа є обов'язковою умовою.

Основною метою таких методів є точне відтворення інформації, що зберігається в часових рядах, в альтернативній математичній структурі, щоб згодом можна було використовувати потужні інструменти теорії графів для характеристики часових рядів з іншої точки зору з метою подолання розриву між нелінійним аналізом часових рядів, динамічних систем і теорією графів.

У даній роботі розглянемо лише алгоритм графа видимості (див. опис у лаб. 11).

14.1.1 Пакет `ts2vg`

Для подальшої побудови класичного VG або його горизонтального аналогу, ми будемо використовувати бібліотеку `ts2vg`. Пакет `ts2vg` надає високопродуктивну реалізацію алгоритму для побудови графів видимості з даних часових рядів, вперше представленого Лукасом Лакасою та ін. [218].

Графи видимості та деякі з їхніх властивостей (наприклад, степеневі розподіли) обчислюються швидко та ефективно навіть для часових рядів з мільйонами спостережень. Для обчислення графів використовується ефективний алгоритм "розділяй і володарюй", коли це можливо [220].

14.1.1.1 Встановлення

Остання випущена версія `ts2vg` доступна на PyPI і може бути легко встановлена шляхом запуску наступної команди:

```
!pip install ts2vg
```

14.1.1.2 Підтримувані типи графів

14.1.1.2.1 Основні типи

1. Класичний граф видимості [218] (`ts2vg.NaturalVG`).
2. Горизонтальний граф видимості [185] (`ts2vg.HorizontalVG`).

14.1.1.2.2 Доступні варіації

1. Зважений граф видимості (через параметр `weighted`).
2. Направлений граф видимості (через параметр `directed`).
3. Параметричний граф видимості [221] (через параметри `min_weight` та `max_weight`).
4. Граф обмеженої проникної видимості [222,223] (через параметр `penetrable_limit`).

Зверніть увагу, що кілька варіантів графів можна комбінувати і використовувати одночасно. Із більш детальною документацією можна ознайомитись на сайті бібліотеки `ts2vg`.

14.1.1.2.3 Сумісність з іншими бібліотеками

Отримані графи можуть бути легко перетворені в графові об’єкти з інших поширених графових бібліотек Python, таких як `igraph`, `NetworkX` та `SNAP` для подальшого аналізу.

Для цього передбачені наступні методи:

- `as_igraph()`;
- `as_network()`;
- `as_snap()`.

14.2 Хід роботи

Спочатку імпортуємо необхідні модулі для подальшої роботи:

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
import neurokit2 as nk
import yfinance as yf
import pandas as pd
import networkx as nx
import scienceplots

from sklearn import preprocessing
from tqdm import tqdm
```

```
from ts2vg import NaturalVG, HorizontalVG
from scipy.spatial import distance

%matplotlib inline
```

І виконаємо налаштування рисунків для виведення:

```
plt.style.use(['science', 'notebook', 'grid']) # стиль, що використовуватиметься
# для виведення рисунків

size = 16
params = {
'figure.figsize': (8, 6),            # встановлюємо ширину та висоту рисунків за
замовчуванням
'font.size': size,                   # розмір фонтів рисунку
'lines.linewidth': 2,                # товщина ліній
'axes.titlesize': 'small',           # розмір титулки над рисунком
'axes.labelsize': size,              # розмір підписів по осям
'legend.fontsize': size,             # розмір легенди
'xtick.labelsize': size,             # розмір розмітки по осі Ox
'ytick.labelsize': size,             # розмір розмітки по осі Ox
"font.family": "Serif",              # сімейство стилів підписів
"font.serif": ["Times New Roman"],   # стиль підпису
'savefig.dpi': 300,                  # якість збережених зображень
'axes.grid': False                   # побудова сітки на самому рисунку
}

plt.rcParams.update(params)          # оновлення стилю згідно налаштувань
```

Розглянемо можливість використання графодинамічних показників у якості індикаторів або індикаторів-передвісників кризових явищ. Для прикладу завантажимо часовий ряд криптовалютного індексу Біткоїна за весь період до 1-го грудня 2023 року, що надається веб-ресурсом Yahoo! Finance:

```
symbol = 'BTC-USD'                       # символ індексу
end = '2023-12-01'                       # кінцевий період
data = yf.download(symbol, end=end)      # вивантажуємо дані
time_ser = data['Adj Close'].copy()      # зберігаємо саме ціни закриття
date_in_num = mdates.date2num(time_ser.index)

xlabel = 'time, days'                    # підпис по вісі Ox
ylabel = symbol                          # підпис по вісі Oy
```

> ⚠ Увага
>
> Виконайте цей блок, якщо хочете зчитати дані не з Yahoo! Finance, а із власного файлу. Зрозуміло, що й аналіз результатів, і висновки залежать від того, з яким рядом ми працюємо

```
symbol = 'sMpa11'                    # Символ індексу
```

```
path = "databases\sMpa11.txt"       # шлях зчитування файлу
data = pd.read_csv(path,            # зчитування даних
                   names=[symbol])
time_ser = data[symbol].copy()      # копіюємо значення до окремої змінної

date_in_num = mdates.date2num(time_ser.index)

xlabel = r'$\varepsilon$'           # підпис по вісі Ox
ylabel = symbol                     # підпис по вісі Oy
```

Виведемо досліджуваний ряд:

```
fig, ax = plt.subplots()                    # Створюємо порожній графік
ax.plot(time_ser.index, time_ser.values)    # Додаємо дані до графіка
ax.legend([symbol])                         # Додаємо легенду
ax.set_xlabel(xlabel)                       # Встановимо підпис по вісі Ox
ax.set_ylabel(ylabel)                       # Встановимо підпис по вісі Oy

plt.xticks(rotation=45)                     # оберт позначок по осі Ox на 45
градусів

plt.savefig(f'{symbol}.jpg')                # Зберігаємо графік
plt.show();                                 # Виводимо графік
```

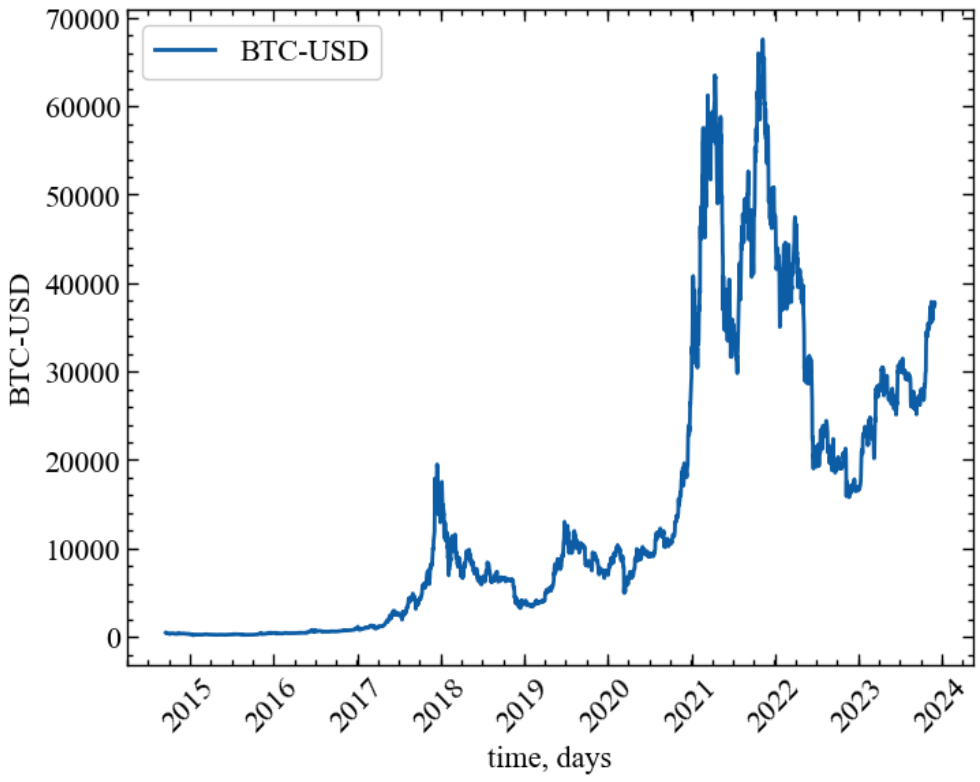


Рис. 14.1: Динаміка щоденних значень індексу Біткоїна

Як і до цього, визначимо функцію для перетворення ряду (його стандартизації або знаходження прибутковостей). Для цього оголосимо функцію `transformation()`, що прийматиме на вхід часовий сигнал, тип ряду, і

повертатиме його перетворення. Як показували попередні дослідження авторів, вихідне представлення часового ряду надає найбільш інформативне представлення для побудови графа. Тим не менш, ми допускаємо, що, наприклад, прибутковості фізичного сигналу можуть мати краще графове представлення, тому і визначаємо цю функцію в даній роботі.

```
def transformation(signal, ret_type):

    for_graph = signal.copy()

    if ret_type == 1:       # Зважаючи на вид ряду, виконуємо
# необхідні перетворення
       pass
    elif ret_type == 2:
       for_graph = for_graph.diff()
    elif ret_type == 3:
       for_graph = for_graph.pct_change()
    elif ret_type == 4:
       for_graph = for_graph.pct_change()
       for_graph -= for_graph.mean()
       for_graph /= for_graph.std()
    elif ret_type == 5:
       for_graph = for_graph.pct_change()
       for_graph -= for_graph.mean()
       for_graph /= for_graph.std()
       for_graph = for_graph.abs()
    elif ret_type == 6:
       for_graph -= for_graph.mean()
       for_graph /= for_graph.std()

    for_graph = for_graph.dropna().values

    return for_graph
```

Повертаємо той самий вихідний сигнал.

Далі задаємо параметри досліджуваного графа. Для подальших розрахунків ми будемо використовувати одні й ті ж самі значення часового вікна, кроку й типу 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_graph = transformation(signal, ret_type) # перетворення сигналу

window = 250           # розмір вікна
tstep = 1              # крок вікна
graph_type = 'classic' # тип графу: classic, horizontal
```

```
length = len(time_ser)
```

14.2.1 Побудова графа

Оскільки побудова графа для всього часового ряду може зайняти досить великий проміжок часу, ми будемо будувати граф видимості лише для його фрагменту. Для цього визначимо параметри `index_begin` та `index_end`, які будуть вказувати на початок відліку побудови та кінець. Для класичного графа видимості маємо:

```
index_begin = 1700
index_end = 2800

date = date_in_num[index_begin:index_end]

if graph_type == 'classic':
    g = NaturalVG(directed=None).build(for_graph[index_begin:index_end],
xs=date)
    pos1 = g.node_positions()
    nxg = g.as_networkx()
if graph_type == 'horizontal':
    g = HorizontalVG(directed=None).build(for_graph[index_begin:index_end],
xs=date)
    pos1 = g.node_positions()
    nxg = g.as_networkx()

graph_plot_options = {
'with_labels': False,
'node_size': 0,
'node_color': [(0, 0, 0, 1)],
'edge_color': [(0, 0, 0, 0.15)],
}

fig, ax = plt.subplots(1, 2, figsize=(15, 8))

nx.draw_networkx(nxg, ax=ax[0], pos=pos1, **graph_plot_options)
ax[0].tick_params(bottom=True, labelbottom=True)
ax[0].plot(time_ser.index[index_begin:index_end],
for_graph[index_begin:index_end], label=fr"{ylabel}")
ax[0].set_title(f'Зв\'язки видимості для {ylabel}', pad=10)
ax[0].set_xlabel(xlabel)
ax[0].set_ylabel(f"{ylabel}")
ax[0].legend(loc='upper right')
ax[0].tick_params(axis='x', labelrotation=45)


ax[1].set_title(f'Графове представлення для {symbol}', pad=10)

# визначаємо позицію вузлів на графі
pos2 = nx.spring_layout(nxg, k=0.15, iterations=100)

# розраховуємо ступеневу центральність
```

```
degCent = nx.degree_centrality(nxg)

# створити список розмірів вершин на основі ступеневої центральності
node_sizes = [v*100 for v in degCent.values()]

# кольори вузлів на основі їх ступеневої центральності
node_colors = [v for v in degCent.values()]

# будуємо граф
nx.draw_networkx(nxg, ax=ax[1], pos=pos2,
                 node_size=node_sizes,
                 node_color=node_colors,
                 with_labels=False,
                 cmap=plt.get_cmap('plasma'))

# присвоюємо мінімальне та максимальне значення
# ступеневої центральності для побудови теплової шкали
vmin = np.asarray(list(degCent.values())).min()
vmax = np.asarray(list(degCent.values())).max()

sm = plt.cm.ScalarMappable(cmap=plt.get_cmap('plasma'),
                           norm=plt.Normalize(vmin=vmin, vmax=vmax))
cb = plt.colorbar(sm, ax=ax[1])
cb.set_label('Ступенева центральність')

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", dpi=1000)
```

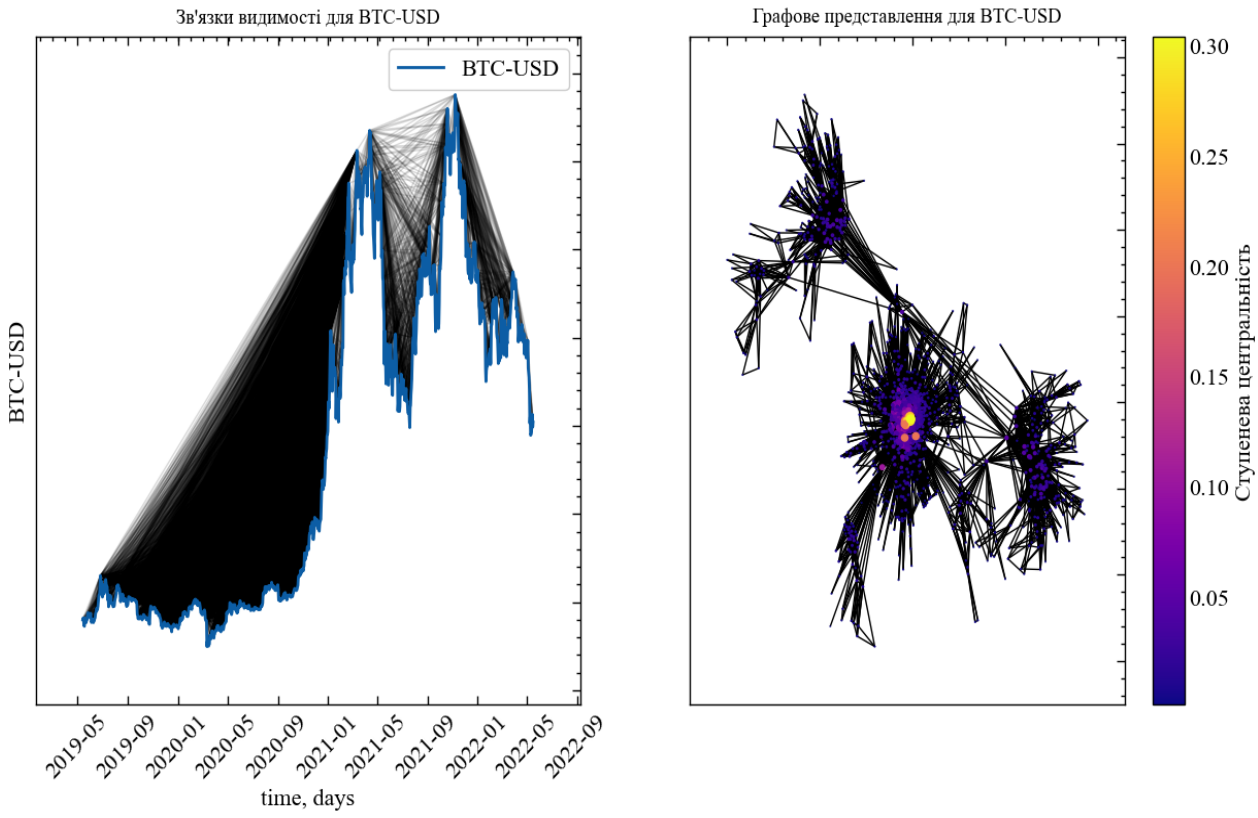


Рис. 14.2: Графік зв’язків видимості на основі природного VG напередодні крахів 21-го року на ринку Біткоїна та графове представлення цього фрагмента

Як ми можемо бачити з представленого рисунку, три послідовних зростання та спадання ціни BTC у 2021-2022 роках характеризуються доволі високим ступенем видимості в передкризовий період. Також дані піки утворюють орієнтовно 3 кластери із високою ступеневою центральністю. Крахові події на криптовалютному ринку можна розглядати як графи переважного приєднання, де, можливо, ключову роль у цих підйомах та спадах можуть відігравати один або декілька “китів” ринку, котрі чинять найбільший вплив на ринок і спрямовують вектор уваги всіх трейдерів у тому чи іншому напрямі.

14.2.2 Віконна процедура

Далі будемо спостерігати за тим, як змінюються властивості мережі з плином часу. Для цього використаємо добре знайому нам процедуру рухомого вікна. У рамках цієї процедури дослідимо графодинаміку як спектральних, так і топологічних показників.

Для побудови парної динаміки конкретного індикатора та досліджуваного ряду визначимо функцію `plot_pair`:

```
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', rotation=35, **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();
```

14.2.2.1 Спектральні характеристики

Спектральна теорія графів базується на вивченні властивостей графів через власні значення або власні вектори матриці суміжності A або **матриці Лапласа** (Laplacian matrix) L [224].

Нагадаємо, що стандартна матриця Лапласа для графа G визначається як

$$L = D - A, \tag{14.1}$$

D — діагональна матриця G, де i-ий діагональний елемент є ступенем вершини i в G [225], а A — матриця суміжності G. У цій роботі ми представляємо спектральні характеристики для нормованої матриці Лапласа [226], яка визначається як

$$\hat{L} = D^{-1/2} L D^{-1/2}. \tag{14.2}$$

Якщо λ — власне значення $\hat{L}$, тоді $\lambda \in [0,2]$ [224]; тобто, нормалізуючи матрицю Лапласа, ми нормалізуємо власні значення.

```
AlgebraicCon = []
GraphEnergy = []
SpecMoment_3 = []
SpecRadius = []
SpecGap = []
NaturalConnectivity = []

for i in tqdm(range(0, length-window, tstep)):
# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
```

```
        nxg = g.as_networkx()

# спектр власних значень матриці суміжності
    adj_spectrum = nx.adjacency_spectrum(nxg).real

# сортуємо власні значення в порядку зростання
    sorted_adj_spectrum = np.sort(adj_spectrum)

# розраховуємо алгебраїчну зв'язність
    alg_con = nx.algebraic_connectivity(nxg, normalized=True,
method='tracemin_lu')

# розраховуємо енергію графа
    graph_en = np.sum(np.abs(adj_spectrum))

# розраховуємо спектральний розрив
    spec_gap = sorted_adj_spectrum[-1] - sorted_adj_spectrum[-2]

# розраховуємо спектральний радіус
    spec_rad = np.max(np.abs(adj_spectrum))

# розраховуємо спектральний момент
    spec_mom_3 = np.mean(adj_spectrum **3)

# розраховуємо природню зв'язність
    nat_con = np.log(np.mean(np.exp(adj_spectrum)))

    AlgebraicCon.append(alg_con)
    GraphEnergy.append(graph_en)
    SpecRadius.append(spec_rad)
    SpecGap.append(spec_gap)
    SpecMoment_3.append(spec_mom_3)
    NaturalConnectivity.append(nat_con)
```

Зберігаємо абсолютні значення у текстовому документі. Також готуємо мітки для рисунків та назви збережених мір:

```
ind_names = ['algebraic_conn', 'graph_energy', 'spectral_radius',
'spectral_grap', 'spectral_moment_3', 'natural_connectivity']

indicators = [AlgebraicCon, GraphEnergy, SpecRadius,
              SpecGap, SpecMoment_3, NaturalConnectivity]

measure_labels = [r'$\lambda_2$', r'$E$', r'$R$', r'$\delta$', r'$m_3$',
r'$N_c$']

file_names = []

for i in range(len(ind_names)):
    name =
f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_seriestype={ret_type
}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

14.2.2.1.1 Алгебраїчна зв'язність

Щодо власних значень матриці Лапласа, однією з основних характеристик, яку ми можемо отримати, є **алгебраїчна зв'язність** (algebraic connectivity) λ_2 графа, яка відповідає другому найменшому власному значенню матриці. Цей показник відображає кількість роз'єднаних компонент. Для незв'язного графа λ_2 буде дорівнювати нулю, а для графа з вищою щільністю зв'язків λ_2 буде більшим. Використовуючи цей показник, можливо визначити відмовостійкість і синхронізованість досліджуваної системи.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="magenta")
```

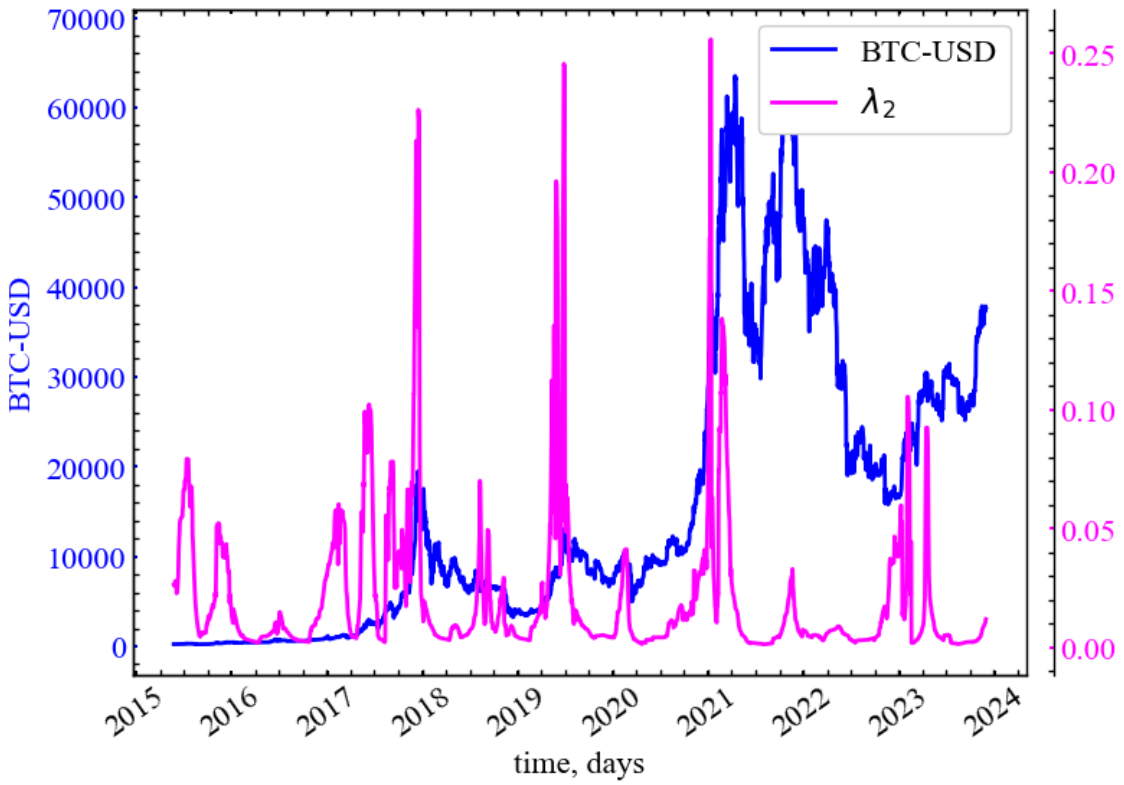


Рис. 14.3: Динаміка індексу BTC та алгебраїчної зв'язності

На Рис. 14.3 видно, що λ_2 зростає в передкризові періоди часу, що говорить про зростання ступеня синхронізованності між трейдерами ринку в дані періоди часу. Мережа крипторинку набуває все більшої корельованості та стійкості. Подібна динаміка може вказувати на зростання узгодженності між великими гравцями ринку щодо своїх подальших дій на Біткоїні.

14.2.2.1.2 Енергія графа

З власних значень матриці суміжності A з G можна визначити таку міру, як **енергія графа** (graph energy) $E(G)$ [227,228]:

$$E = E(G) = \sum_{i=1}^{N} |\lambda_i|. \tag{14.3}$$

Подібно до λ_2, ми маємо повністю роз'єднаний граф, коли $E(G) = 0$. Для кожного $\lambda_i > 0$ існує багато ребер e_{ij}, які визначають високу та ефективну зв'язність G.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="crimson")
```

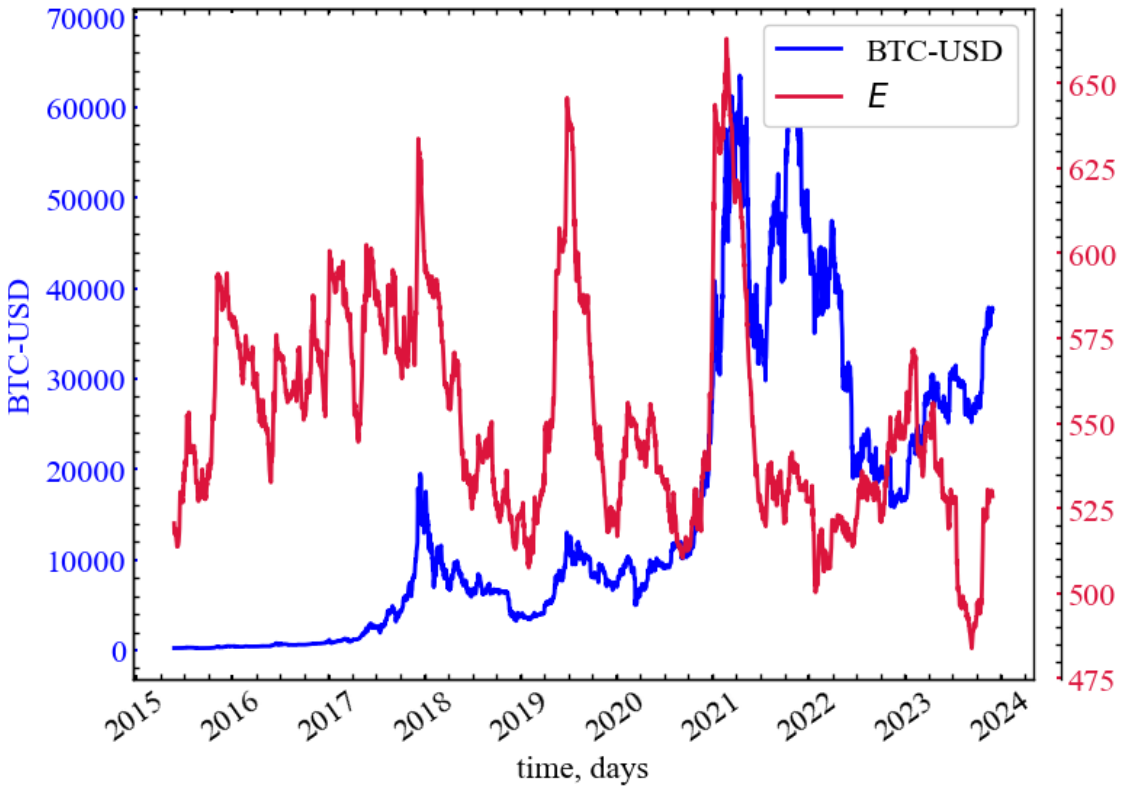


Рис. 14.4: Динаміка індексу BTC та енергії графа

Рис. 14.4 демонструє, що в періоди відносної стабільності E залишається на досить низькому рівні, що вказує на роз'єднаність трейдерів ринку в подібні періоди. Як покупці, так і продавці діють досить некорельовано. У передкризові

періоди енергія починає зростати, що вказує на зростання ефективності роботи між гравцями ринку та їх зв'язності.

14.2.2.1.3 Спектральний радіус

Крім наведених вище мір, можна визначити такі міри, як **спектральний радіус** (spectral radius), яка є найбільшим абсолютним власним значенням матриці A:

$$R = R(G) = \max_{1 \le i \le N} |\lambda_i|. \tag{14.4}$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[2],
          ylabel,
          measure_labels[2],
          xlabel,
          file_names[2],
          clr="orange")
```

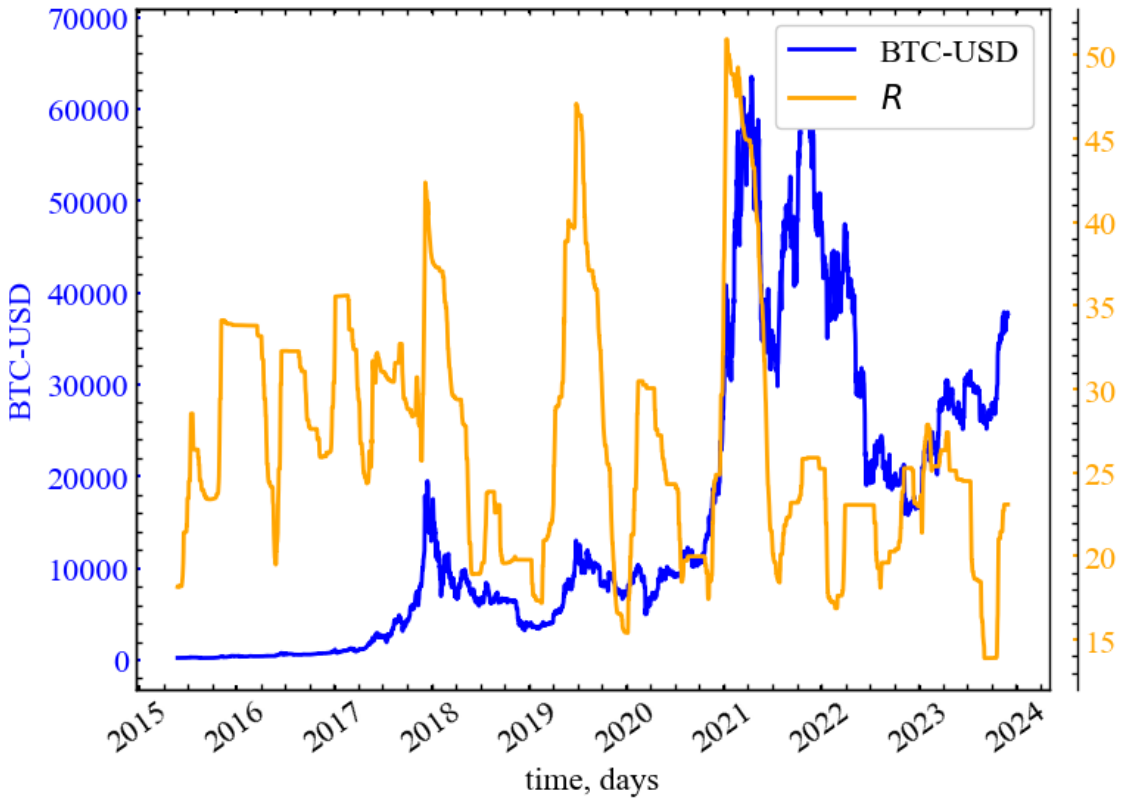


Рис. 14.5: Динаміка індексу BTC та спектрального радіуса

На даному рисунку (Рис. 14.5) видно, що спектральний радіус зростає в кризові й передкризові періоди, що вказує на зростання корельованості графа Біткоїна та синхронізованості дій трейдерів.

14.2.2.1.4 Спектральний розрив

Проранжувавши власні значення матриці суміжності G у неспадаючому порядку, тобто $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, ми можемо визначити таку міру, як **спектральний розрив** (spectral gap):

$$\delta = \delta(G) = \lambda_n - \lambda_{n-1} \tag{14.5}$$

для якого λ_n — перше найбільше власне значення $\hat{L}$, а λ_{n-1} друге найбільше власне значення. Спектральний розрив показує швидкість синхронізації в досліджуваній мережі. Чим він більший, тим більш взаємопов'язаними є вершини і тим складнішим є граф.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[3],
          ylabel,
          measure_labels[3],
          xlabel,
          file_names[3],
          clr="darkgreen")
```

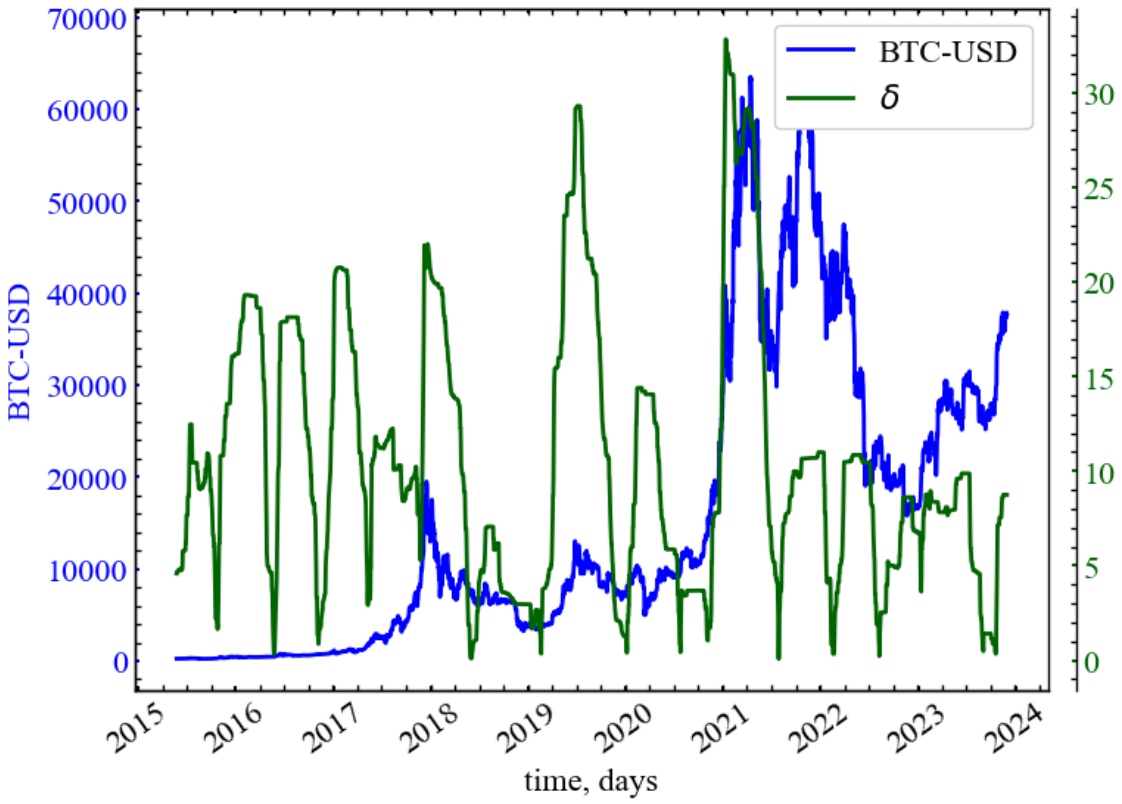


Рис. 14.6: Динаміка індексу BTC та спектрального розриву

Рис. 14.6 демонструє, що спектральний розрив також є показником синхронізованності ринку в передкризові період. Однак, із динаміки даного

показника можна сказати, що в моменти криз найбільшу кількість інформації починає нести найбільше власне значення матриці Лапласа. Можна припустити, що друге і третє також можуть слугувати в якості індикаторів крахових подій, але найбільше власне значення в даному випадку представляється найкращим рішенням.

14.2.2.1.5 Спектральний момент

Спектральною мірою складності, яку ми також хотіли б представити є k**-й спектральний момент** (spectral moment). Для невід'ємного цілого числа k, k-ий спектральний момент визначається як

$$m_k = m_k(G) = \sum_{i=1}^{N} \lambda_i^k, \tag{14.6}$$

де m_k дорівнює кількості замкнутих обходів довжини k [229]. Кількість замкнутих обходів є важливим показником для вимірювання складності системи. Як було показано в роботі By та ін. [230], використовуючи кількість замкнутих обходів всієї довжини, ми можемо виміряти складність графа та надлишковість альтернативних найкоротших шляхів. Отже, більші значення m_k відповідають більшій складності мережі. Для подальших обчислень ми обрали $k = 3$.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[4],
          ylabel,
          measure_labels[4],
          xlabel,
          file_names[4],
          clr="chocolate")
```

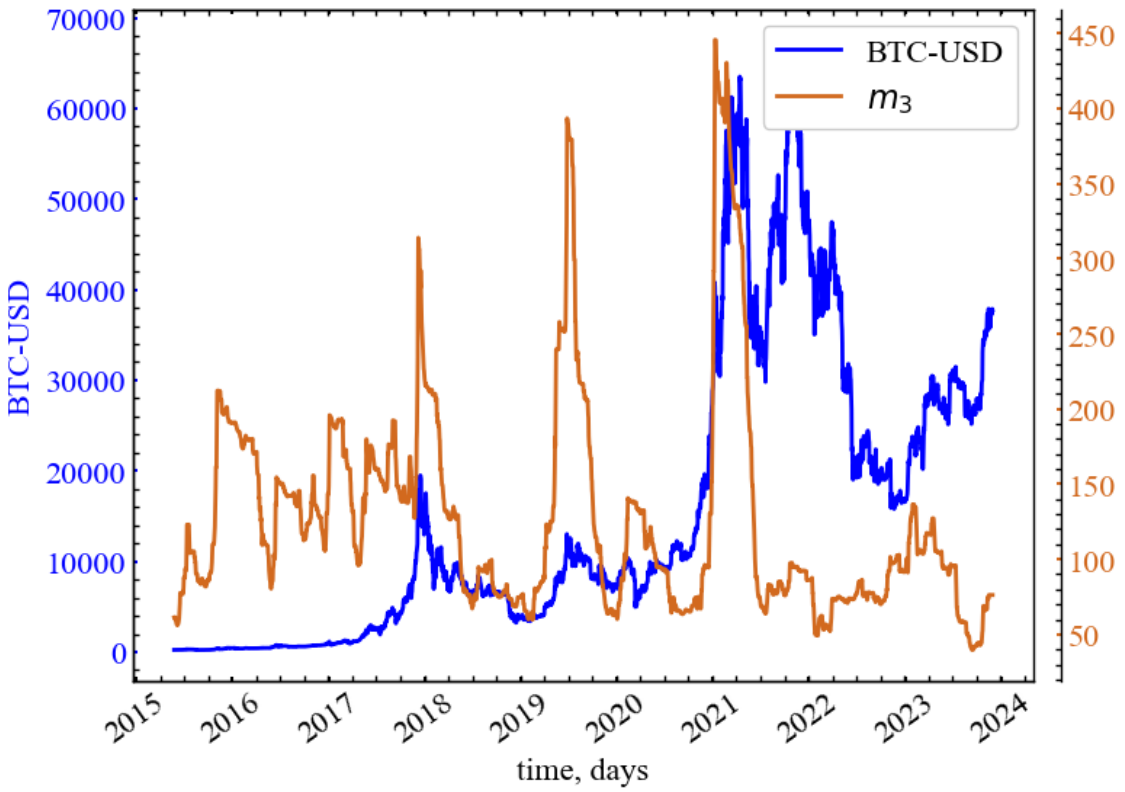


Рис. 14.7: Динаміка індексу BTC та спектрального моменту

З динаміки m^3 видно, що найбільш суттєвим ступенем синхронізованності характеризувалась предкризова динаміка 2018, середини 2019 і 2021 років. У ці періоди часу ми мали найбільшу кількість досить високих власних значень матриці Лапласа, а одже й достатньо високий ступінь синхронізації ринку в зазначені періоди.

14.2.2.1.6 Спектральна природна зв'язність

Юнь та ін. [231] запропонували вимірювати "середнє власне значення" спектра суміжності графа G. Було запропоновано називати цей показник **природною зв'язністю** (natural connectivity) або **природним власним значенням**:

$$N_c = N_c(G) = \ln\left(\frac{1}{N}\sum_{i=1}^{N}\exp\lambda_i\right). \tag{14.7}$$

Естрада [232], Ву та ін. [230] показали, що (14.7) є чутливою та надійною мірою стійкості мережі.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[5],
```

```
        ylabel,
        measure_labels[5],
        xlabel,
        file_names[5],
        clr="black")
```

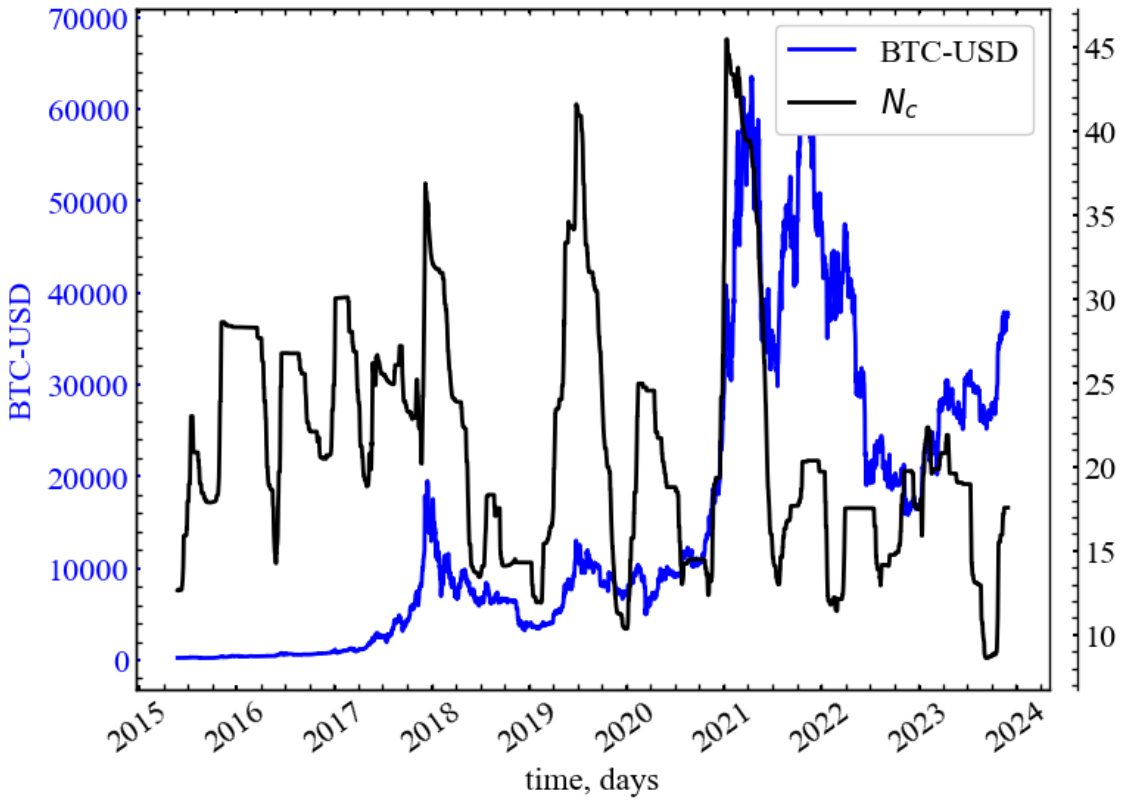


Рис. 14.8: Динаміка індексу BTC та спектральної природної зв'язності

На Рис. 14.8 видно, що показник природної зв'язності зростає у передкризові періоди часу. Тобто, даний показник можна використовувати в якості індикатора або індикатора-передвісника крахових подій на ринку Біткоїна. Особливо характерним є зростання ступеня синхронізованності ринку напередодні 2018 року, що може вказувати на початкові стадії зміцнення стійкості криптовалютної мережі. Досить високий ступінь синхронізованності ринку можна спостерігати напередодні середини 2019 та початку 2021 років.

14.2.2.2 Топологічні міри центральності

Існує багато способів кількісно оцінити важливість вершини або ребра з точки зору певного мережного атрибуту, відображаючи таким чином **топологію** складної мережі.

```
DegreeMax = []
GlobalEigenvectorCentrality = []
GlobalClosenessCentrality = []
GlobalInformationCentrality = []
```

```
GlobalBetweennessCentrality = []
GlobalHarmonicCentrality = []

for i in tqdm(range(0,length-window,tstep)):
# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# максимальний ступінь вершини
    deg_max = max(dict(nxg.degree()).values())

# середній ступінь впливовості
    glob_eigenvector_centrality =
np.mean(list(nx.eigenvector_centrality_numpy(nxg).values()))

# середній ступінь близькості
    glob_closeness_centrality =
np.mean(list(nx.closeness_centrality(nxg).values()))

# середній ступінь інформаційності
    glob_information_centrality =
np.mean(list(nx.information_centrality(nxg).values()))

# максимальний ступінь посередництва
    glob_betweenness_centrality =
np.max(list(nx.betweenness_centrality(nxg).values()))

# середній ступінь гармонійності
    glob_harm_centrality = np.mean(list(nx.harmonic_centrality(nxg).values()))

    DegreeMax.append(deg_max)
    GlobalEigenvectorCentrality.append(glob_eigenvector_centrality)
    GlobalClosenessCentrality.append(glob_closeness_centrality)
    GlobalInformationCentrality.append(glob_information_centrality)
    GlobalBetweennessCentrality.append(glob_betweenness_centrality)
    GlobalHarmonicCentrality.append(glob_harm_centrality)
```

Зберігаємо абсолютні значення у текстовому документі. Також готуємо мітки для рисунків та назви збережених:

```
ind_names = ['DegreeMax', 'GlobalEigenvectorCentrality',
'GlobalClosenessCentrality',
'GlobalInformationCentrality', 'GlobalBetweennessCentrality',
'GlobalHarmonicCentrality']
```

```
indicators = [DegreeMax, GlobalEigenvectorCentrality, GlobalClosenessCentrality,
              GlobalInformationCentrality, GlobalBetweennessCentrality,
GlobalHarmonicCentrality]

measure_labels = [r'$D_{max}$', r'$X$', r'$C$', r'$I$', r'$B$', r'$GHc$']

file_names = []

for i in range(len(ind_names)):
    name =
f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_seriestype={ret_type
}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

14.2.2.2.1 Максимальний ступінь вершини

Ступінь вершини (node degree) або **ступенева центральність** (degree centrality) є концептуально найпростішою метрикою для опису характеристик зв'язку однієї вершини в складній мережі. Вона може бути представлена у вигляді

$$d_i = \sum_{j=1}^{N} A_{ij}, \tag{14.8}$$

де d_i підраховує кількість j-х ребер, що інцидентні вершині i.

Окрім ступеня конкретної вершини, ми можемо визначити вершину з найбільшою кількістю інцидентних ребер. Кількість таких вершин можемо позначити як D_{max}:

$$D_{max} = \max_{i=1,\dots,N} d_i. \tag{14.9}$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="magenta")
```

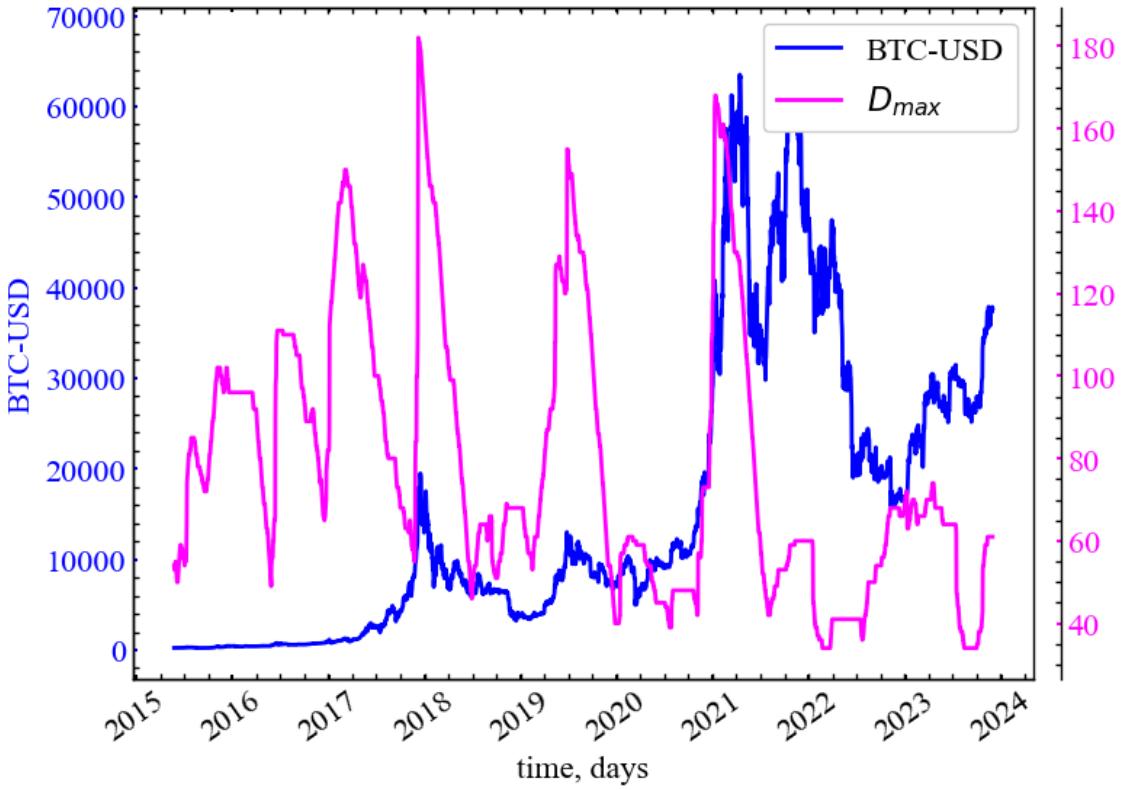


Рис. 14.9: Динаміка індексу BTC та максимального ступеня вершини

На даному рисунку (Рис. 14.9) видно, що максимальний ступінь вершини починає зростати в кризові та передкризові періоди, що вказує на зростання центральності одного або декількох вузлів. Можна припустити, що один або декілька трейдерів ринку починають концентрувати на собі увагу всіх інших діячів дотичних до ринку Біткоїна.

14.2.2.2.2 Середній ступінь впливовості

Ступінь впливовості (eigenvector centrality) обчислює оцінку важливості вузла шляхом додавання впливовостей його сусідів. Впливовість для вузла i — це i-й елемент власного вектора x, пов'язаний з власним значенням λ максимального модуля, який є додатним. Такий власний вектор x визначається з точністю до мультиплікативної константи рівнянням

$$\lambda x^T = x^T A, \tag{14.10}$$

де A — матриця суміжності графа G. Наведене вище рівняння еквівалентне наступному:

$$\lambda x^T = \sum_{j \to i} x_j, \tag{14.11}$$

Тобто, додавання ступенів впливовості попередників вершини i дає ступінь впливовості i, помножену на λ. У випадку неорієнтованих графів x також розв'язує знайоме рівняння $Ax = \lambda x$.

За теоремою Перрона-Фробеніуса [233], якщо G сильно зв'язний, то існує єдиний власний вектор x, і всі його елементи строго додатні.

Якщо G не є сильно зв'язним, то може існувати декілька лівих власних векторів, пов'язаних з λ, причому деякі з їх елементів можуть дорівнювати нулю.

> ⓘ Примітка
>
> Ступінь впливовості або центральність за власним вектором було введено Ландау [234] для шахових турнірів. Пізніше його знову відкрив Вей [235], а потім популяризував Кендалл [236] в контексті спортивного рейтингу. Берге ввів загальне визначення для графів, заснованих на соціальних зв'язках [237]. Бонасіч [238] знову ввів центральність власного вектора і зробив її популярною в аналізі зв'язків.
>
> Ця функція обчислює лівий домінуючий власний вектор, що відповідає додаванню впливовості попередників: це звичайний підхід. Щоб додати центральність наступників, спочатку переверніть граф за допомогою `G.reverse()`.
>
> Ця реалізація використовує SciPy sparse eigenvalue solver (ARPACK) для пошуку найбільшої пари власне значення/власний вектор за допомогою ітерацій Арнольді [239]

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="crimson")
```

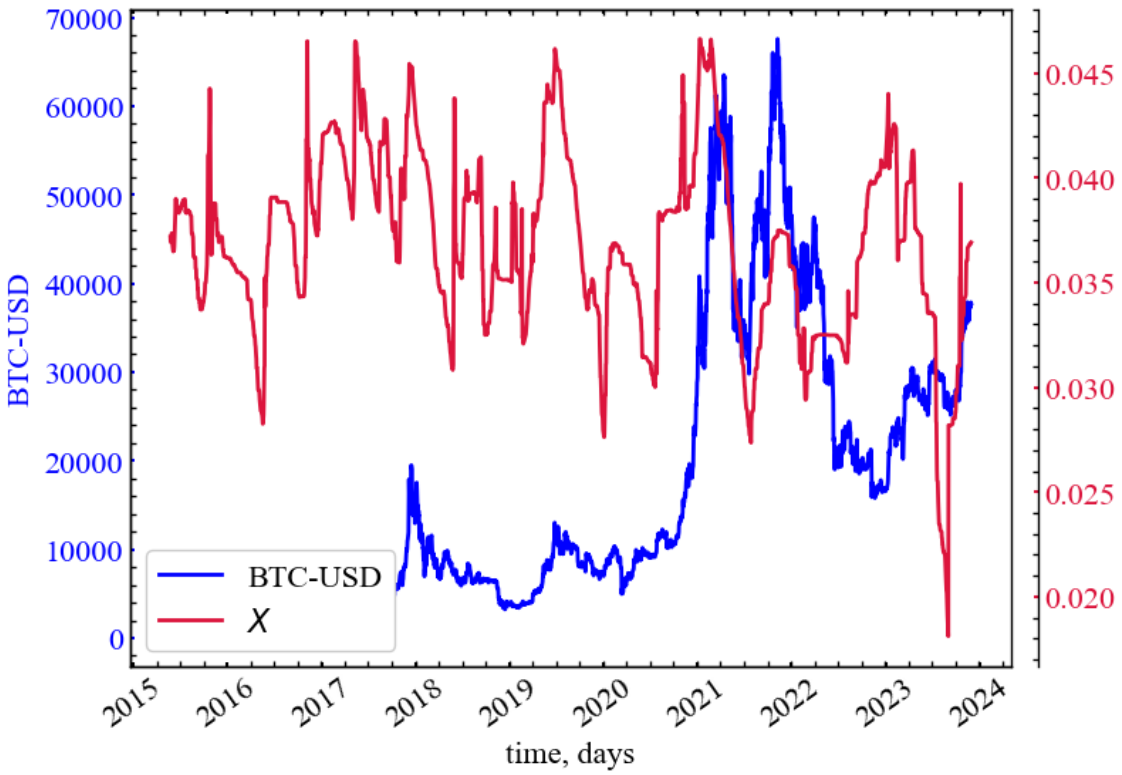


Рис. 14.10: Динаміка індексу BTC та середнього ступеня впливовості

Середній ступінь впливовості характеризується зростанням у передкризові періоди, як правило, спадом під час криз. Подібно до спектральних показників, середній ступінь впливовсті вказує на зростання одного або декількох власних значень матриці суміжності графа та значущості власних векторів центральних вузлів торгівельного графа Біткоїна.

14.2.2.2.3 Середній ступінь близькості

У мережі відстань l_{ij} між вузлом i та вузлом j позначає кількість ребер, які з'єднують найкоротший шлях між цими двома вузлами. Спираючись на поняття довжини найкоротшого шляху між двома вузлами, ми можемо надати різні міри, які характеризують зв'язність всієї мережі. Однією з таких мір є **центральність** (centrality) або **середній ступінь близькості** (average degree connectivity) зв'язку між вершиною i та всіма іншими вершинами

$$c_i = (N-1) \Big/ \left(\sum_{j=1}^{N} l_{ij} \right), \tag{14.12}$$

що надають зворотнє середнє по всім найкоротшим шляхам від i до всіх вузлів j.

Середнє арифметичне значення ступеня близькості для кожного i-го вузла дає нам **глобальний (середній) ступінь близькості**:

$$C = \frac{1}{N}\sum_{i=1}^{N} c_i. \tag{14.13}$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[2],
          ylabel,
          measure_labels[2],
          xlabel,
          file_names[2],
          clr="orange")
```

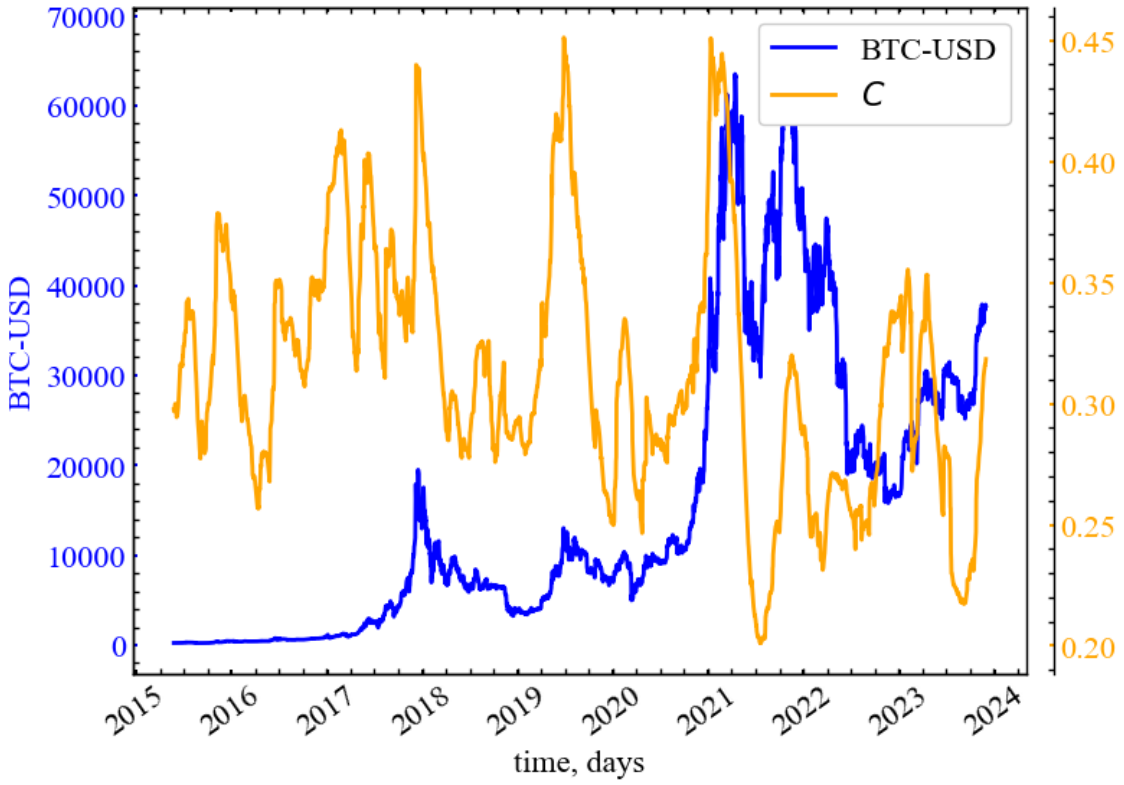


Рис. 14.11: Динаміка індексу BTC та середнього ступеня близькості

Як можна бачити з представленого рисунку (Рис. 14.11), глобальний ступінь близькості зростає в кризові та передкризові періоди, що вказує на спад довжини найкоротших шляхів у графі видимості криптовалютного ринку. Даний показник зростав напередодні останніх чотирьох років. Це вказує на те, що передодні досліджуваних торгівельних років між трейдерами зростав ступінь синхронізованності їх дій.

14.2.2.2.4 Середній ступінь інформаційності

Для визначення центральності будь-якого вузла i пропонується спочатку визначити його інформаційну зв'язність з іншими вузлами, тобто $\{I_{ij}|j = 1,\dots,N\}$. Середнє гармонійне значення інформації щодо шляху від вузла i до інших вузлів буде використовуватися для визначення **ступеня інформаційності** (information centrality) вузла i. Зокрема, якщо I_i пов'язано з центральністю або інформаційністю вузла i, то

$$\widehat{I_i} = \left(\frac{1}{N}\sum_{j=1}^{N}\frac{1}{I_{ij}}\right)^{-1}. \tag{14.14}$$

Згідно зі Стівенсоном та Зеленом [240], ступінь інформаційності можна обчислити шляхом інвертування простої матриці. Перш за все, ми визначаємо $N \times N$ матрицю $B = \{b_{ij}\}$, де

$$b_{ij} = \begin{cases} 0 & \text{якщо } i \text{ та } j \text{ суміжні,} \\ 1 & \text{інакше,} \end{cases} \tag{14.15}$$

і $b_{ii} = 1 + d_i$, де d_i ступінь вершини i.

Далі, визначивши матрицю $C = \{c_{ij}\} = B^{-1}$, ми можемо розрахувати I_{ij} згідно рівняння

$$I_{ij} = \left(c_{ii} + c_{jj} - 2c_{ij}\right)^{-1}. \tag{14.16}$$

Елемент $\sum_{j=1}^{N} 1/I_{ij}$ у рівнянні (14.14) можна переписати наступним чином:

$$\sum_{j=1}^{N} c_{ii} + c_{jj} - 2c_{ij} = Nc_{ii} + T - 2R, \tag{14.17}$$

де $T = \sum_{j=1}^{N} c_{jj}$ і $R = \sum_{j=1}^{N} c_{ij}$.

Отже, ступінь інформаційності вузла i може бути представлений як

$$I_i = [(Nc_{ii} + T - 2R)/N]^{-1} = [c_{ii} + (T - 2R)/N]^{-1}. \tag{14.18}$$

Схожим чином, для вимірювання глобального ступеня інформаційності ми розглядаємо середнє арифметичне локального ступеня інформаційності:

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N} I_i. \tag{14.19}$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[3],
          ylabel,
          measure_labels[3],
          xlabel,
          file_names[3],
          clr="darkgreen")
```

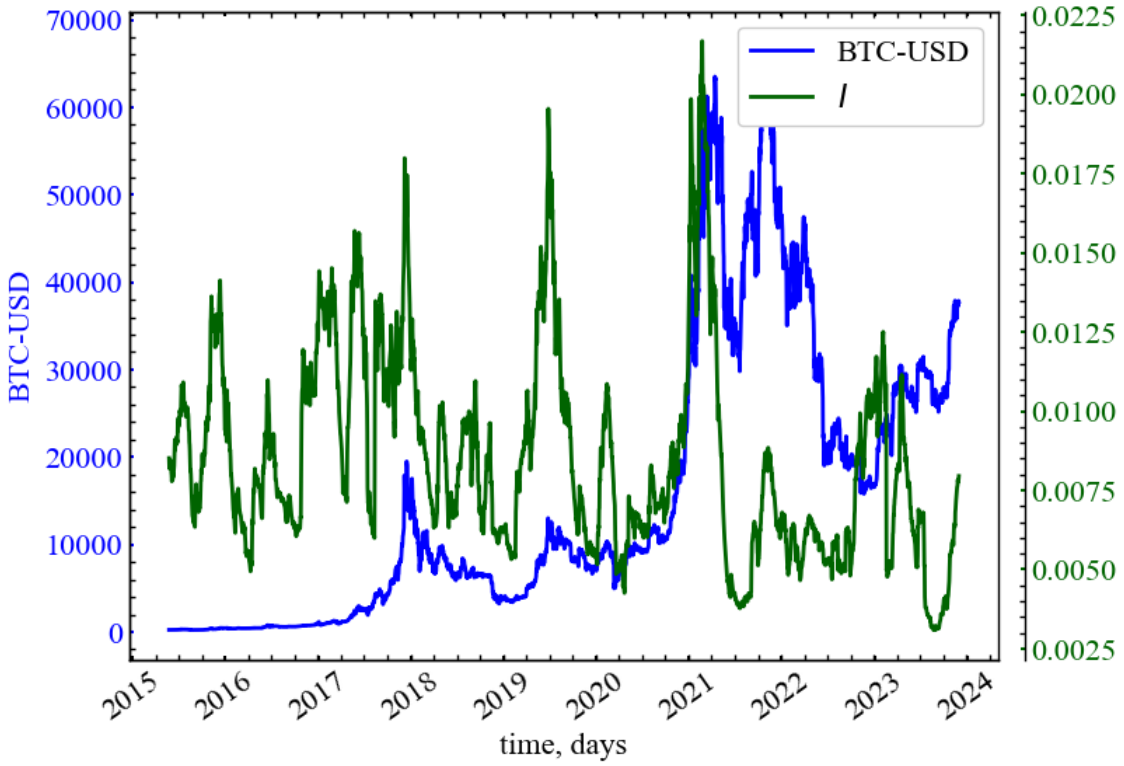


Рис. 14.12: Динаміка індексу BTC та середнього ступеня впливовості

На (Рис. 14.12) видно, що глобальний ступінь інформаційності зростає в передкризові періоди, що є індикатором зростання ефективності передачі інформації між трейдерами ринку та зростання детермінованності динаміки ринку.

14.2.2.2.5 Максимальний ступінь посередництва

Іншою часто досліджуваною характеристикою вершин на основі шляхів є **ступінь посередництва** (betweenness centrality), яка вимірює частку всіх найкоротших шляхів у мережі, що проходять від i до j через вершину k. Для загальної кількості найкоротших шляхів між вершинами i та j, позначених як

$\sigma(i,j)$, і найкоротших шляхів, що проходять через дану вершину $k\big(\sigma(i,j|k)\big)$, ступінь посередництва можна визначити як

$$b_k = \sum_{i,j=1;i,j\neq k}^{N} \sigma\,(i,j|k)/\sigma(i,j). \tag{14.20}$$

Щоб віднайти найбільшу кількість інформації, що проходить через конкретний k-й, ми вимірюємо максимальний ступінь посередництва, розглядаючи кожен k-ий вузол:

$$B = \max_{i=1,\dots,N} b_i. \tag{14.21}$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[4],
          ylabel,
          measure_labels[4],
          xlabel,
          file_names[4],
          clr="chocolate")
```

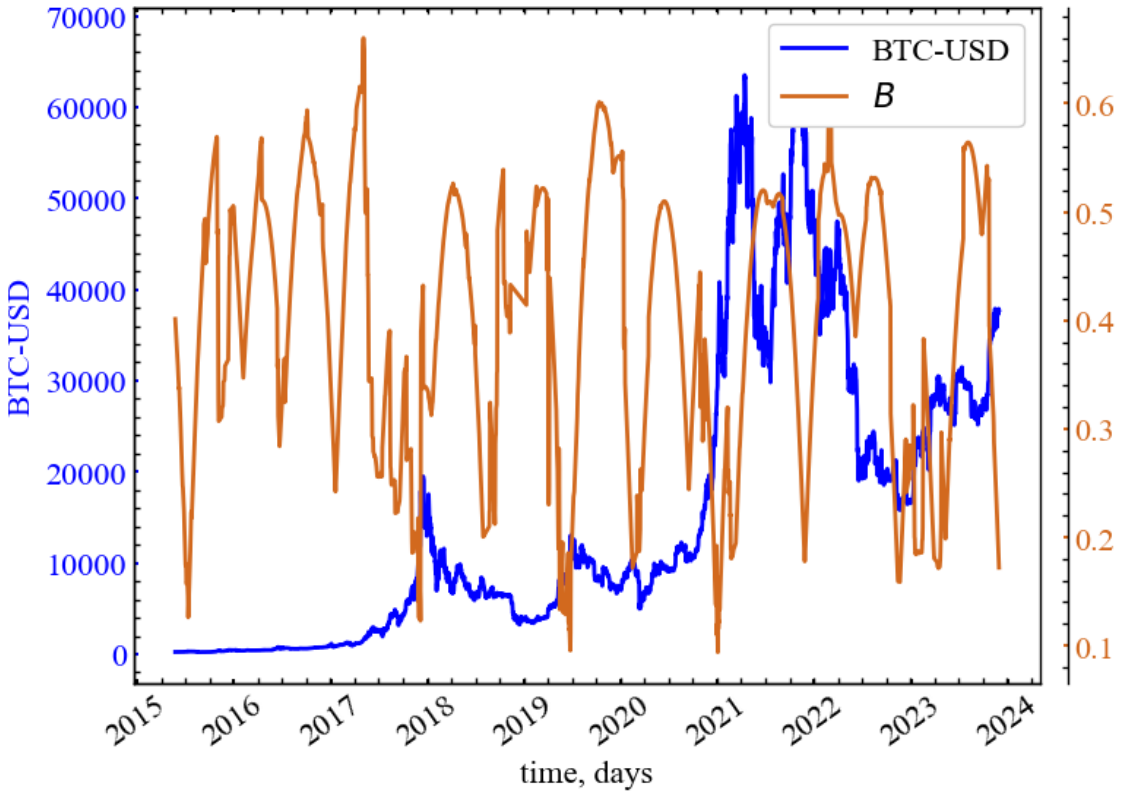


Рис. 14.13: Динаміка індексу BTC та середнього ступеня посередництва

Як можна бачити, показник максимального ступеня посередництва спадає в передкризові періоди, що вказує на спад кількості посередників через яких може

проходити інформація стосовно подальної динаміки Біткоїна. Це говорить про те, що на ринку зв'являються один або декілька трейдерів на котрих концентрується увага майже всіх інших, і зв'язок усіх трейдерів із найбільш впливомими може здійснюватись в один або декілька найкоротших шляхів.

14.2.2.2.6 Середній ступінь гармонійності

Марчіорі та Латора [241] запропонували міру, подібну до (14.12), яка називається **ступенем гармонійності** (harmonic centrality). Для заданого вузла j вона може бути визначена як

$$Hc_j = \sum_{i=1, i\neq j}^{N} \left(l_{ij}\right)^{-1}, \tag{14.22}$$

де $\left(l_{ij}\right)^{-1} = 0$, якщо між вузлами i та j немає шляху. Середній ступінь гармонійності визначається через середнє арифметичне локальних ступенів гармонійності.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[5],
          ylabel,
          measure_labels[5],
          xlabel,
          file_names[5],
          clr="black")
```

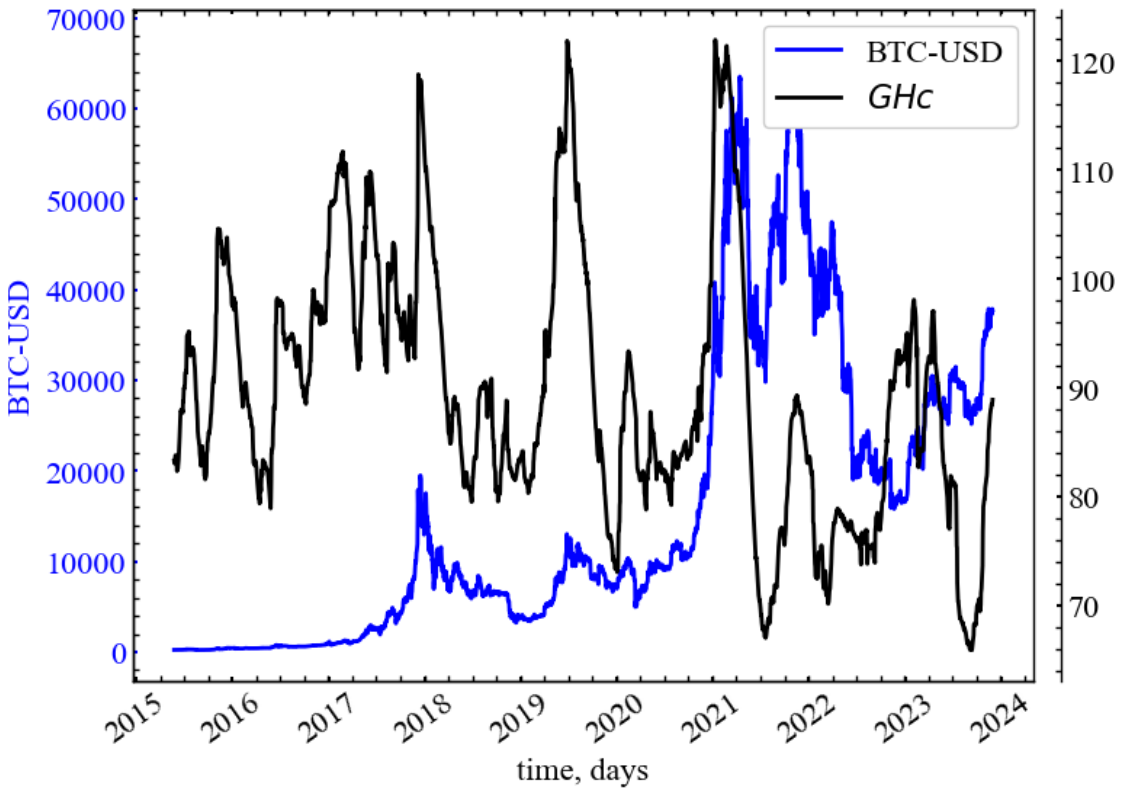


Рис. 14.14: Динаміка індексу BTC та середнього ступеня гармонійності

14.2.2.3 Асортативність

Асортативність (Assortativity) означає тенденцію в мережі до з'єднання вузлів з подібними властивостями, тоді як диассортативність виявляється у з'єднанні вузлів з різнорідними властивостями. Реальні мережі можуть демонструвати різну асортативність. Соціальні мережі, такі як взаємодії між вченими або корпоративними директорами, зазвичай мають позитивну асортативність. З іншого боку, технологічні та біологічні мережі, такі як електромережі, Інтернет, білкові взаємодії, нейронні мережі та харчові мережі, зазвичай виявляють негативну асортативність.

Далі буде представлено декілька показників асортативності для передчасної ідентифікації криптовалютних криз.

```
Assortativity = []
AvgDegreeConnectivity = []

for i in tqdm(range(0, length-window, tstep)):
# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
```

```
        g = NaturalVG(directed='left_to_right').build(fragm)
        pos = g.node_positions()
        nxg_dir = g.as_networkx()
     if graph_type == 'horizontal':
        g = HorizontalVG(directed='left_to_right').build(fragm)
        pos = g.node_positions()
        nxg_dir = g.as_networkx()

# розрахунок асортативності
    assort = nx.degree_pearson_correlation_coefficient(nxg_dir)

# середня степенева зв'язність
    avg_deg_con = np.mean(list(nx.average_degree_connectivity(nxg_dir,
source="in", target="in").values()))

    Assortativity.append(assort)
    AvgDegreeConnectivity.append(avg_deg_con)

ind_names = ['Assortativity', 'AvgDegreeConnectivity']

indicators = [Assortativity, AvgDegreeConnectivity]

measure_labels = [r'$r$', r'$\langle d_{nn}^{w} \rangle$']

file_names = []

for i in range(len(ind_names)):
    name =
f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_seriestype={ret_type
}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

14.2.2.3.1 Середня ступенева зв'язність

Середня ступенева зв'язність (average degree connectivity) $d_{nn}(d)$ для вершин зі ступенем d є ще однією мірою, яка використовується для дослідження структури мереж [242]. Оскільки вона може бути виражена як $d_{nn}(d) = \sum_{d'} P(d'|d)$, де $P(d'|d)$ — умовна ймовірність того, що дана вершина зі ступенем d пов'язана з вершиною зі ступенем d'. Ця величина виражає кореляцію між ступенями зв'язаних вершин [243]. За відсутності кореляцій між ступенями, $P(d'|d)$ не залежить від d, а також від середнього ступеня найближчих сусідів, тобто $d_{nn}(d) = \text{const}$ [242]. За наявності кореляцій поведінка $d_{nn}(d)$ визначає два загальні класи мереж. Якщо $d_{nn}(d)$ є зростаючою функцією від d, тоді вершини з високим (низьким) ступенем мають більшу ймовірність бути пов'язаними з вершинами з вищим (нижчим) ступенем. Ця властивість у різних галузях науки називається *асортативним змішуванням* [244]. Навпаки, спадна поведінка $d_{nn}(d)$ визначає *дизасортативне змішування*,

в тому сенсі, що вершини з високим (низьким) ступенем мають більшість сусідів з низьким (високим) ступенем вершин.

Міру такої сортативності чи дизасортативності для сусідів певної вершини i можна визначити як середню ступеневу зв'язність (середньозважений ступінь найближчого сусіда):

$$d_i^w = \frac{1}{s_i} \sum_{j=1}^{N} A_{ij}\, w_{ij} d_j, \tag{14.23}$$

де $s_i = \sum_{j=1}^{N} A_{ij}\, w_{ij}$ — це "сила" i-го вузла; A_{ij} — це елемент матриці суміжності A; w_{ij} — це вага ребра e_{ij} (у нашому випадку вона дорівнює 1); d_j представляє ступінь вершини j-го сусіда.

Загалом, це рівняння вимірює ступінь тяжіння сусідів з високим або низьким ступенем вершини один до одного відносно величини фактичних взаємодій.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="darkorange")
```

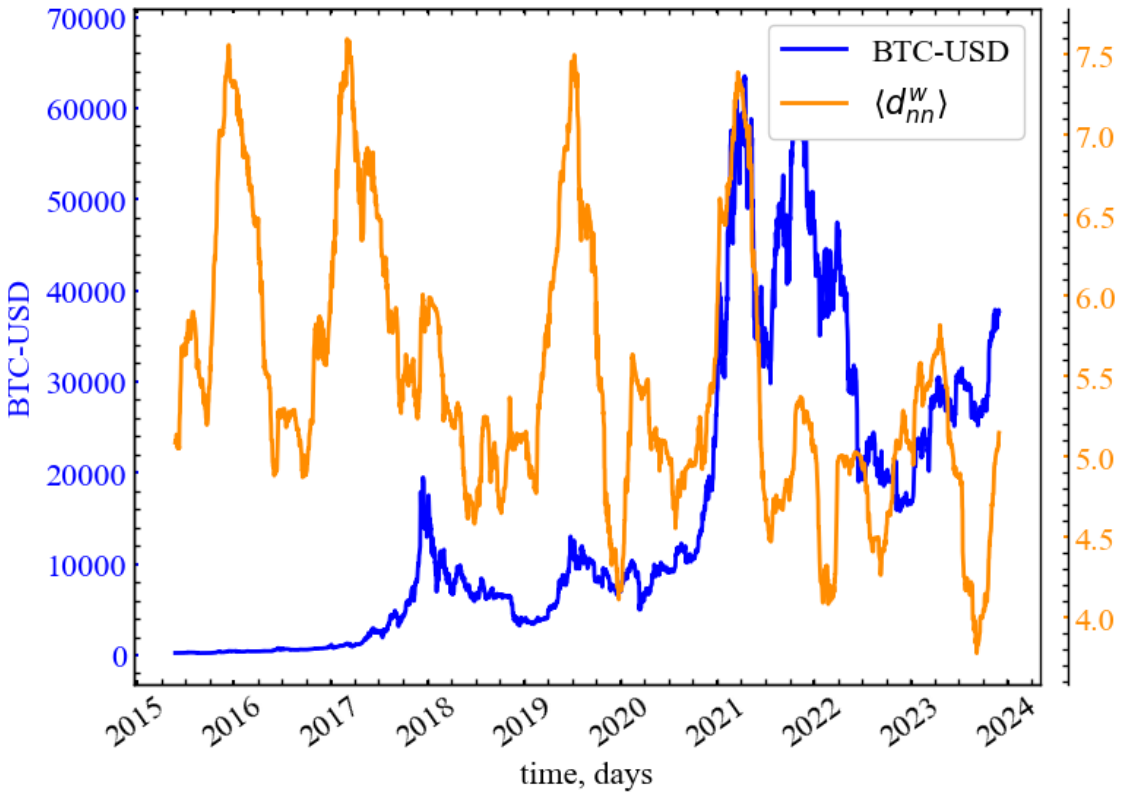


Рис. 14.15: Динаміка індексу BTC та середньої ступеневої зв’язності

Як можна бачити з даного рисунка (Рис. 14.15), середня ступенева зв’язність зростає в передкризові періоди, що вказує на поступове зростання ступеня тяготіння вершин з високою степеневою центральністю до вершин із ще вищою центральністю.

14.2.2.3.2 Ступінь асортативності

Інша форма асортативного змішування залежить від однієї або декількох скалярних властивостей вершин мережі. Для його обчислення ми визначаємо матрицю e_{ij}, яка задовольняє правилам додавання: $\sum_{ij} e_{ij} = 1$, $\sum_j e_{ij} = a_i, = a_i$, $\sum_i e_{ij} = b_j$, де a_i та b_j — частки ребер, які починаються та закінчуються у вершинах i та j. Розрахувавши коефіцієнт кореляції Пірсона, можна визначити **ступінь асортативності** (degree of assortativity) [244]. Таким чином, цей коефіцієнт асортативності обчислюється як

$$r = \sum_{xy} xy\left(e_{xy} - a_x b_y\right)/\sigma_a \sigma_b, \tag{14.24}$$

а σ_a та σ_b визначають стандартні відхилення розподілів a_x та b_y; $-1 \le r \le 1$, де $r < 0$ вказує на вищу дизасортативність, $r > 0$ демонструє вищу асортативність, а $r = 0$ говорить про відсутність асортативності між вершинами.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="darkgreen")
```

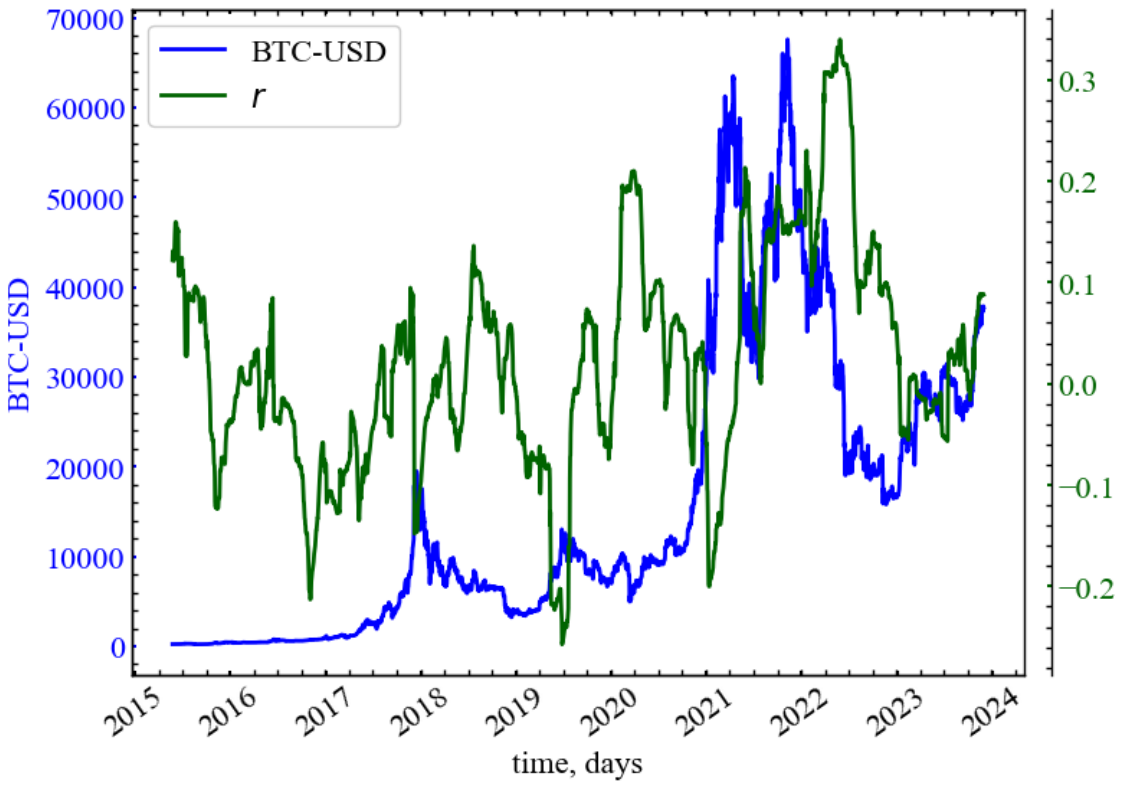


Рис. 14.16: Динаміка індексу BTC та середнього ступеня асортативності

На Рис. 14.16 видно, що коефіцієнт асортативності спадає в передкризові періоди, що вказує на дизасортативну поведінку ринку в ці моменти часу: вершини з малим ступенем зв'язності й централізованості тяжіють до вершин, що характеризуються високим ступенем посередництва, гармонійності, інформаційності, близькості тощо. Як уже зазначалося, дизасортативність, що властива передкризовим періодам Біткоїна, характерна і як для реальних соціальних мереж, так і для складних біологічних мереж.

14.2.2.4 Кластеризація

У теорії графів, коефіцієнт кластеризації вказує на те, наскільки вузли у графі мають тенденцію групуватися. Дослідження показують, що у більшості реальних мереж, зокрема, у соціальних мережах, вузли зазвичай утворюють компактні групи з високою кількістю зв'язків між ними.

Для подальшого аналізу розглянемо показники транзитивності, глобальної тріадної і квадратичної кластеризацій.

```
Transitivity = []
AvgClustering = []
AvgSquareClustering = []

for i in tqdm(range(0, length-window, tstep)):
# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()
```

```
# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# транзитивність
    trans = nx.transitivity(nxg)

# глобальний коефіцієнт кластеризації
    avg_clust = nx.average_clustering(nxg)

# коефіцієнт квадратичної кластеризації
    avg_sqr_clust = np.mean(list(nx.square_clustering(nxg).values()))

    Transitivity.append(trans)
    AvgClustering.append(avg_clust)
    AvgSquareClustering.append(avg_sqr_clust)

ind_names = ['AvgClustering', 'Transitivity', 'AvgSquareClustering']

indicators = [AvgClustering, Transitivity, AvgSquareClustering]

measure_labels = [r'$\langle C_3 \rangle$', r'$T$', r'$\langle C_4 \rangle$']

file_names = []

for i in range(len(ind_names)):
    name =
f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_seriestype={ret_type
}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

14.2.2.4.1 Коефіцієнт глобальної кластеризації

Для того, щоб охарактеризувати щільність зв'язків між сусідами вершини i, ми можемо використати коефіцієнт локальної кластеризації:

$$C_i^3 = \sum_{k,j=1}^{N} A_{ik}A_{kj}A_{ji} \Big/ d_i(d_i - 1), \tag{14.25}$$

де чисельник позначає кількість закритих трикутників, що містять вершину i.

Ми можемо розглядати **глобальний коефіцієнт кластеризації** (global clustering coefficient), як середнє арифметичне локального коефіцієнта кластеризації трикутників [208]:

$$\langle C^3 \rangle = \frac{1}{N} \sum_{i=1}^{N} C_i^3, \tag{14.26}$$

що вимірює середню схильність системи до утворення трикутних кластерів.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="magenta")
```

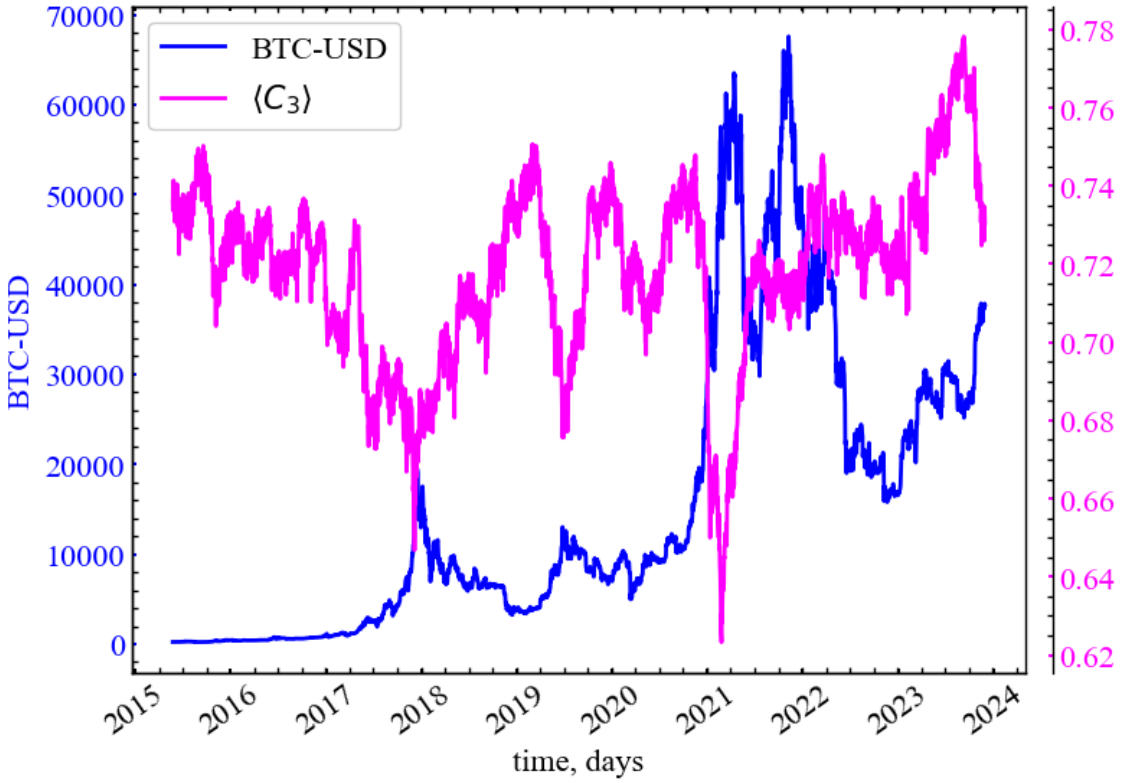


Рис. 14.17: Динаміка індексу BTC та глобального коефіцієнта кластеризації

На Рис. 14.17 видно, що в абсолютних значеннях глобальний коефіцієнт тріадної кластеризації залишається на достатньо високому рівні, що говорить про досить високий ступінь кластеризації трейдерів криптовалютного ринку. Локально, в передкризові періоди, видно, що $\langle C^3 \rangle$ спадає, що говорить про локалізовану руйнацію кластеризованих групувань трейдерів і зростання їх тяжіння до одного або декількох гравців ринку.

14.2.2.4.2 Транзитивність

У випадку дуже неоднорідних степенів, тобто безмасштабних мереж, де лише кілька вершин мають високу степінь, а інші — низьку ($d_i < 2$), вершини з низькою степенню будуть брати участь переважно в обчисленні локального коефіцієнта кластеризації, що може призвести до недооцінки трикутних кластерів у мережі. Баррат і Вайгт [245] запропонували альтернативний підхід для подолання такої проблеми, який отримав назву **транзитивності** (transitivity) [246]:

$$T = \sum_{k,j=1}^{N} A_{ik}\, A_{kj} A_{ji} \Big/ \sum_{i,k,j=1}^{N} A_{ik}\, A_{ji}. \qquad (14.27)$$

У реальних мережах ми можемо зіткнутися з випадками, коли зв'язані сусіди в мережі можуть утворювати різні кліки (форми кластеризації). Класичний коефіцієнт локальної кластеризації, який вимірює імовірність знаходження трикутників, зазвичай відповідає одностороннім мережам. Однак він не може бути сформований у двосторонніх мережах [247,248]. Складні структури односторонніх, двосторонніх і багатосторонніх мереж реальної системи можуть призвести до утворення кластерів набагато вищого порядку.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="crimson")
```

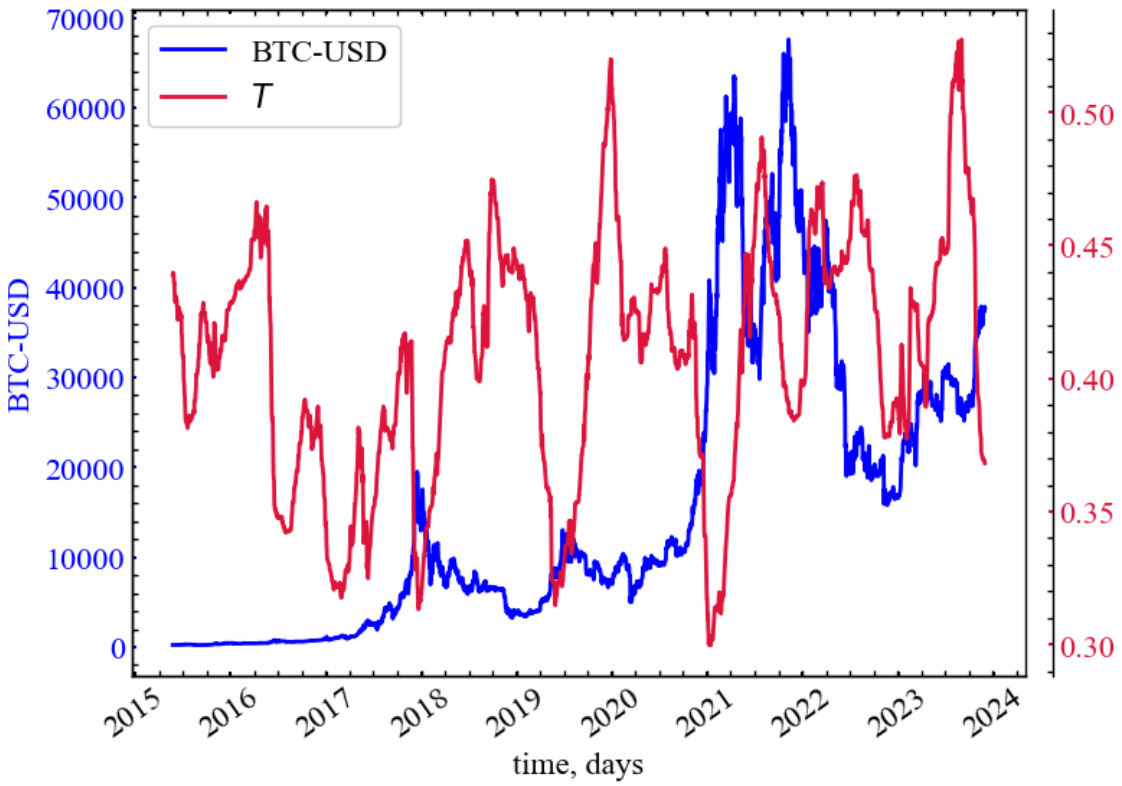


Рис. 14.18: Динаміка індексу BTC і транзитивності

Показник транзитивності працює подібно до $\langle C^3 \rangle$. Однак, на відміну від $\langle C^3 \rangle$, він надає куди більше сигналів про подальшу крахову поведінку на ринку Біткоїна. Видно, що на ринку зберігається досить висока частка трикутних кліків, які стають неповними в передкризові періоди, на що і вказує спадання T.

14.2.2.4.3 Коефіцієнт квадратичної кластеризації

Подібно до C_i^3, який є класичним коефіцієнтом локальної кластеризації, було запропоновано кількісно оцінити коефіцієнт кластеризації C_i^4 [249], який відповідає ймовірності знайти "квадратний" кластер, утворений сусідами вузла i. Тобто, що два сусіди вузла i мають спільного сусіда, відмінного від i. Для кожної вершини i вона може бути обчислена як

$$C_i^4 = \frac{\sum_{k=1}^{d_i} \sum_{j=k+1}^{d_i} q_i\,(k,j)}{\sum_{k=1}^{d_i} \sum_{j=k+1}^{d_i} [a_i(k,j) + q_i(k,j)]}, \tag{14.28}$$

де $q_i(k,j)$ представляє кількість спостережуваних квадратних кластерів; $a_i(k,j) = \big(d_k - (1 + q_i(k,j) + \theta_{ki})\big) + \Big(d_j - \big(1 + q_i(k,j) + \theta_{kj}\big)\Big)$; $\theta_{kj} = 1$ якщо k і j є зв'язними і 0 у зворотньому випадку [250]. Схожим чином до (14.26) ми

можемо визначити **глобальний коефіцієнт квадратичної кластеризації** (global square clustering coefficient) як

$$\langle C^4 \rangle = \frac{1}{N} \sum_{i=1}^{N} C_i^4. \tag{14.29}$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[2],
          ylabel,
          measure_labels[2],
          xlabel,
          file_names[2],
          clr="orange")
```

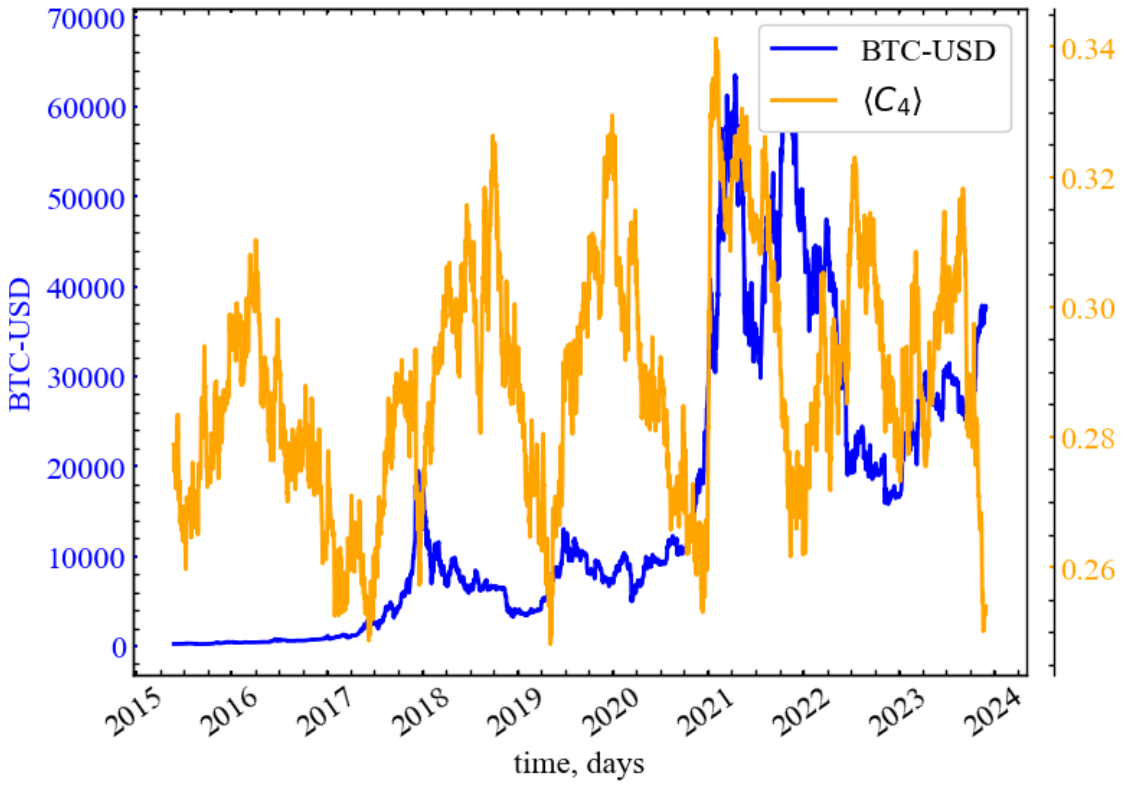


Рис. 14.19: Динаміка індексу BTC і коефіцієнта квадратичної кластеризації

Рис. 14.19 демонструє, що глобально Біткоїн містить куди меншу частку квадратичних кластерів у порівнянні з тріадними. Локально ми спостерігаємо подібну до попередніх показників динаміку: $\langle C_4 \rangle$ спадає в передкризовий період і поступово зростає в посткризовий. Можна зробити таке саме припущення, що й до цього: у передкризові періоди трейдери починають поступово ізолюватися від аналітики один одного і спрямовувати свою увагу на дії одного або декількох найбільш впливових груп. Хоча їх кластеризація спадає, але дії залишаються узгодженими згідно тієї інформації, що доходить до них із зовні.

14.2.2.5 Зв’язність

У математиці **зв’язний граф** — це граф, у якого кількість ребер наближається до максимально можливої (коли кожна пара вершин з’єднана одним ребром). І навпаки, **розріджений граф** містить лише невелику кількість ребер. Точне визначення того, який граф вважати зв’язним або розрідженим, є неоднозначним. Отже, визначення щільності графа може змінюватись у залежності від контексту задачі.

```
Density = []

for i in tqdm(range(0, length-window, tstep)):
# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# розрахунок щільності
    dens = nx.density(nxg)

    Density.append(dens)

ind_names = ['Density']

indicators = [Density]

measure_labels = [r'$\rho$']

file_names = []

for i in range(len(ind_names)):
    name =
f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_seriestype={ret_type
}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

14.2.2.5.1 Щільність

Щільність (density) графа може допомогти визначити, наскільки густо заселений різними ребрами представлений граф. Чим вона вища, тим більшою є зв'язність досліджуваного графа. Її можна обчислити як

$$\rho = E/E_{max}, \tag{14.30}$$

де E дорівнює кількості ребер у G, а $E_{max} = N(N-1)/2$ — це максимальна кількість ребер у простому ненаправленому графі.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="black")
```

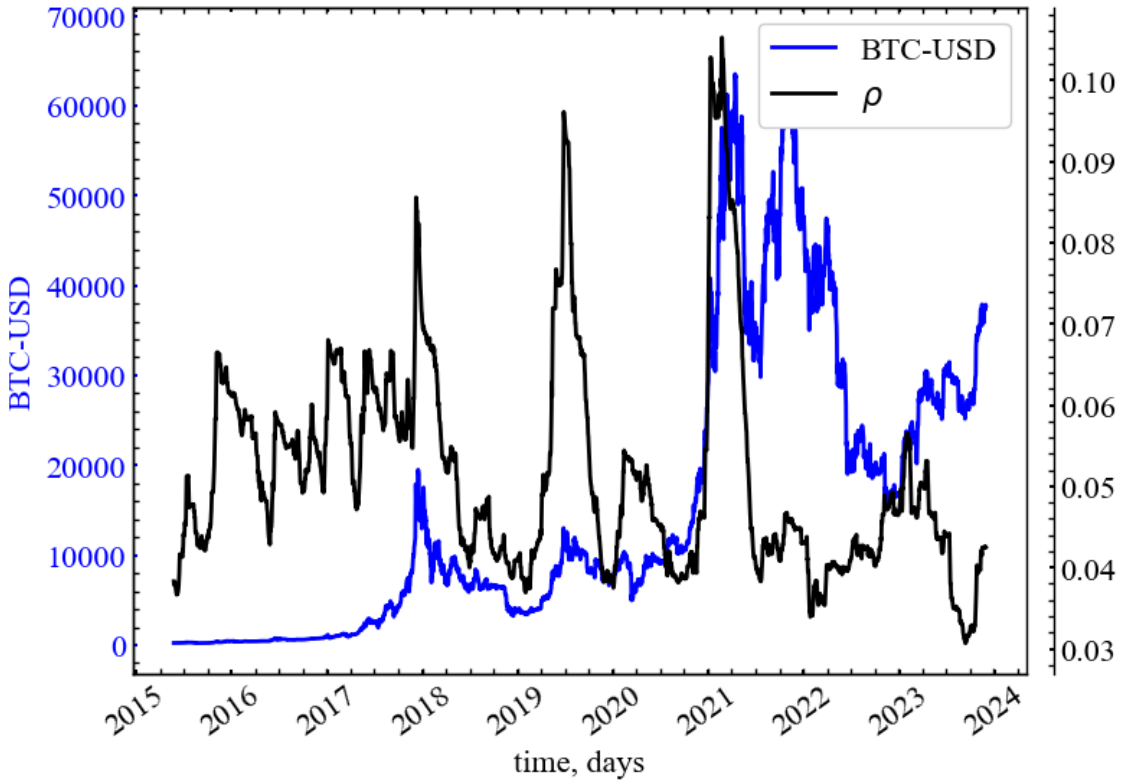


Рис. 14.20: Динаміка індексу BTC і показника щільності

На рисунку можна бачити, що глобальна зв'язність ринку залишається досить низькою ($\rho < 0.10$), що говорить про недостатньо високий рівень зв'язності між теперішніми та минулими вузлами цінових коливань ринку Біткоїна. Віконна динаміка ρ вказує на те, що в передкризовий момент часу

ступінь щільності зв’язків учасників ринку зростає, що робить граф Біткоїна більш стійким.

14.2.2.6 Міри відстані

На основі довжини найкоротшого шляху графа ми можемо отримати безліч інших показників його ефективності або віддаленості його вершин від центру зв’язності досліджуваного графа.

```
Diameter = []
Radius = []

for i in tqdm(range(0, length-window, tstep)):
# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# розрахунок діаметра
    diameter = nx.diameter(nxg)

# розрахунок радіуса
    rad = nx.radius(nxg)

    Diameter.append(diameter)
    Radius.append(rad)

ind_names = ['Diameter', 'Radius']

indicators = [Diameter, Radius]

measure_labels = [r'$diam$', r'rad']

file_names = []

for i in range(len(ind_names)):
    name =
f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_seriestype={ret_type
}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

14.2.2.6.1 Діаметр

Зауважимо, що найкоротший шлях, який є характеристикою відстані між досліджуваними вершинами i та j, може бути використаний для характеристики загального розміру мережі. Величина, яка визначає найбільшу відстань між вершиною i та будь-якою іншою вершиною, називається **ексцентриситетом** (eccentricity):

$$\varepsilon(i) = \max_j l_{ij}. \tag{14.31}$$

Розмір мережі можна охарактеризувати в термінах **діаметру** (diameter) і визначити як

$$diam = \max_i \varepsilon(i) = \max_i \max_j l_{ij}. \tag{14.32}$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="magenta")
```

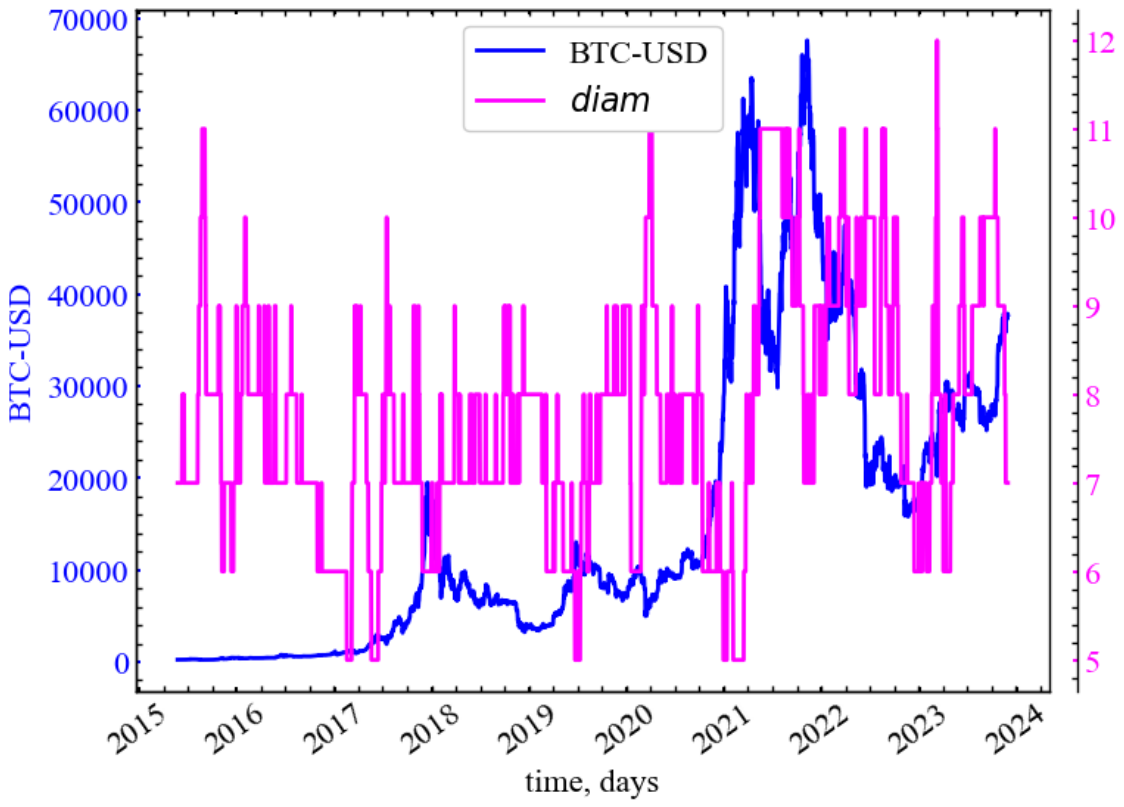


Рис. 14.21: Динаміка індексу BTC і діаметра мережі

На Рис. 14.21 видно, що діаметр графа спадає в передкризовий період, що говорить про зближення верхньої границі графа до його центру. Тобто, інформація, що проходитиме на криптовалютному ринку від одного трейдера до іншого, займатиме набагато менше кроків. Іншими словами, у передкризові періоди трейдери все менше покладаються на посередників із різноманітних новинних ресурсів і більше часу відводять на пряме опрацювання торгівельних закономірностей на ринку.

14.2.2.6.2 Радіус

Таким чином, діаметр — це найбільша (максимальна) довжина шляху в мережі. Отже, ми можемо визначити найменший ексцентриситет досліджуваної мережі, який називається **радіусом** (radius):

$$rad = \max_i \varepsilon(i) = \min_i \max_j l_{ij}. \tag{14.33}$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="crimson")
```

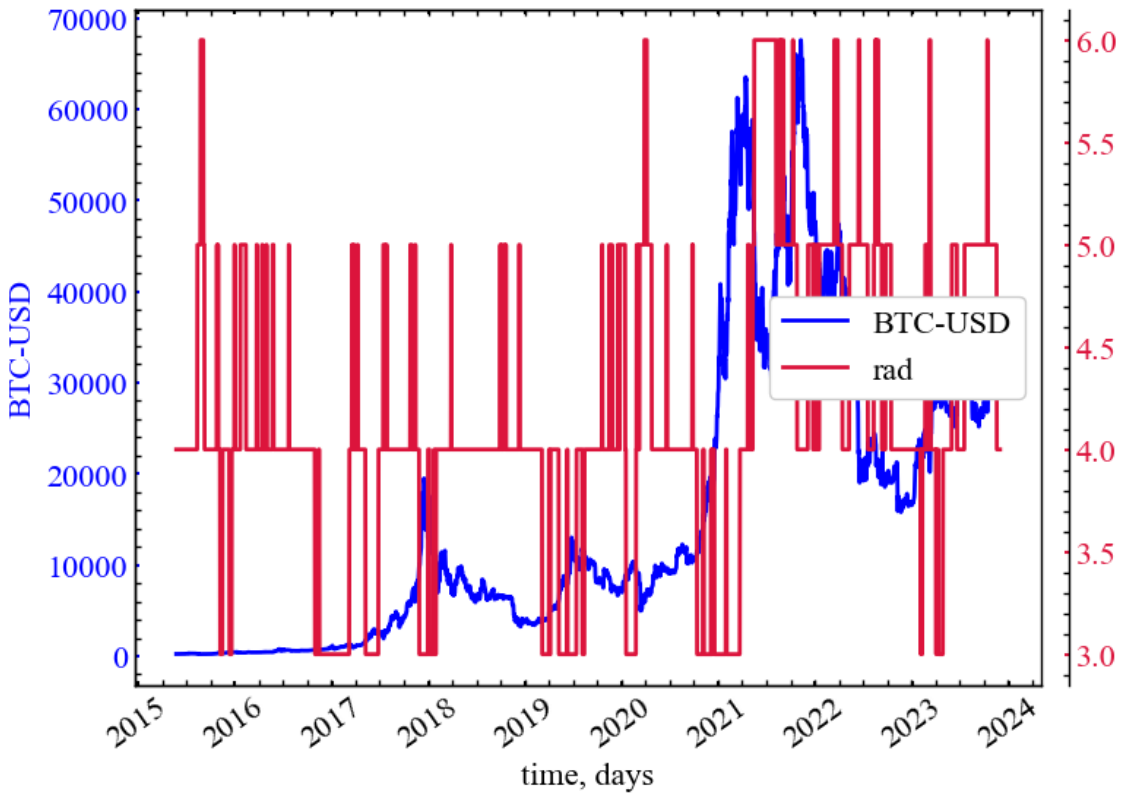


Рис. 14.22: Динаміка індексу BTC і радіуса мережі

Оскільки радіус графа це найменший ексцентриситет мережі, а діаметр є найбільшим, можна зробити подібний висновок. Якщо придивитися, то можна помітити, що радіус представляє приблизно в двічі меншу за діаметр версію, але тренд цих обох індикаторів ідентичний.

14.2.2.7 Ефективність

У галузі мережної науки **ефективність мережі** (network efficiency) в передачі інформації, яку також називають комунікаційною ефективністю, є ключовою метрикою. Це поняття ґрунтується на припущенні, що чим далі один від одного знаходяться два вузли в мережі, тим менш ефективною стає їхня комунікація. Ефективність можна аналізувати як на локальному, так і на глобальному рівнях мережі. На глобальному рівні оцінюється загальний обмін інформацією по всій мережі, де інформаційні потоки протікають паралельно. На локальному рівні вимірюється стійкість мережі до збоїв у менших масштабах. Зокрема, локальна ефективність вузла і відображає, наскільки ефективно його сусіди обмінюються інформацією за його відсутності.

```
LocalEfficiency = []
GlobalEfficiency = []

for i in tqdm(range(0, length-window, tstep)):
# відбираємо фрагмент
```

```
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# розрахунок локальної ефективності
    local_eff = nx.local_efficiency(nxg)

# розрахунок глобальної ефективності
    glob_eff = nx.global_efficiency(nxg)

    LocalEfficiency.append(local_eff)
    GlobalEfficiency.append(glob_eff)

ind_names = ['LocalEfficiency', 'GlobalEfficiency']

indicators = [LocalEfficiency, GlobalEfficiency]

measure_labels = [r'$E_{loc}$', r'$E_{glob}$']

file_names = []

for i in range(len(ind_names)):
    name =
f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_seriestype={ret_type
}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

14.2.2.7.1 Глобальна ефективність

Визначення поведінки в малому світі згідно з [241] можна подати в термінах ефективності E мережі. Ефективність ε_{ij} між вершинами i та j визначається як $1/l_{ij}$. Коли $l_{ij} = \infty$ і, послідовно, якщо $1/l_{ij} = 0$, i і j вважаються роз'єднаними. Відповідно до формалізму ефективності, вона може бути кількісно визначена як для глобальних, так і для локальних масштабів G. Латора та Марчіорі підкреслювали, що $1/L$ та C можна розглядати як перші наближення **глобальної** (E_{glob}) та **локальної** (E_{loc}) **ефективності**.

Середню (глобальну) ефективність G можна визначити як

$$E_{glob} = \sum_{i,j=1} \left(l_{ij}\right)^{-1} / N(N-1). \tag{14.34}$$

Для найбільш ефективного графа, де інформація поширюється найбільш ефективно, E_{glob} набуває максимального значення, а в іншому випадку — мінімального.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[1],
          ylabel,
          measure_labels[1],
          xlabel,
          file_names[1],
          clr="indigo")
```

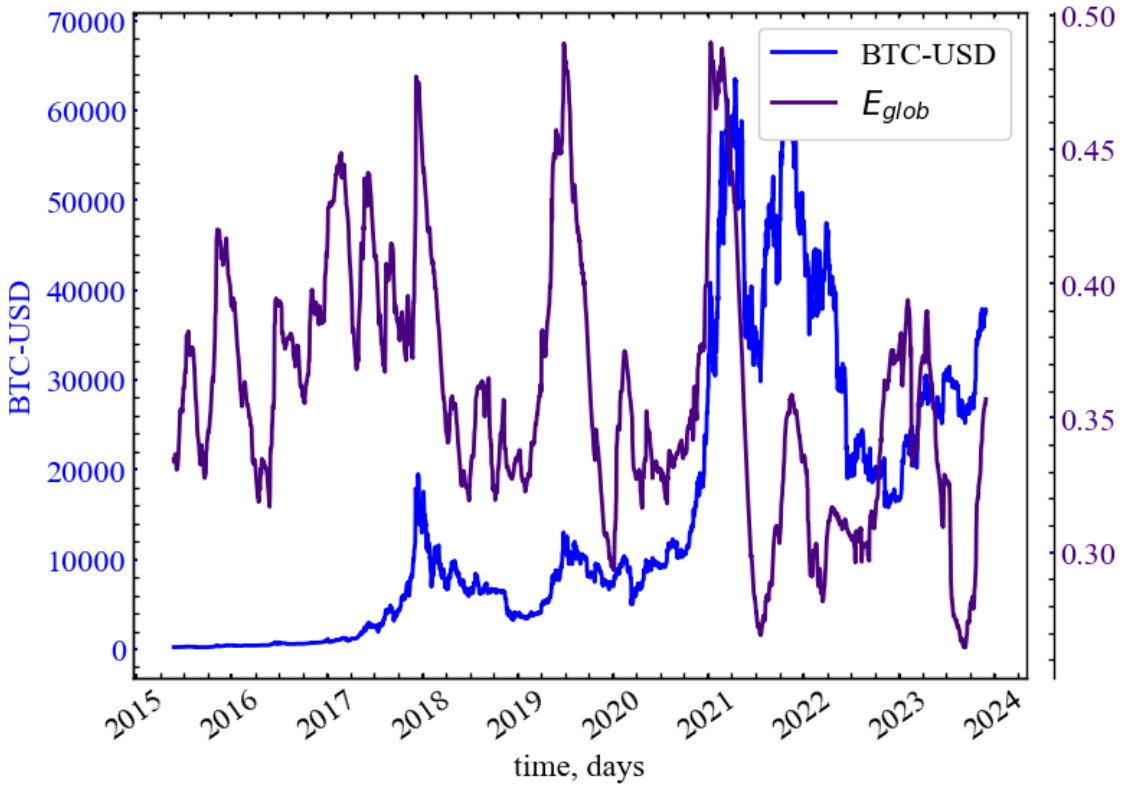


Рис. 14.23: Динаміка індексу BTC і глобальної ефективності мережі

На Рис. 14.23 видно, що ступінь глобальної ефективності мережі зростає в передкризові періоди, що вказує на зростання ступеня проходження інформації в мережі. З точки зору графа видимості, Біткоїн починає діяти в більш детермінований спосіб, де зв'язність його графа видимості стає близькою до топології ідеального графа, де вся інформація передається в найефективніший спосіб.

14.2.2.7.2 Локальна ефективність

Локальна ефективність відіграє роль, подібну до глобального коефіцієнта кластеризації. Локальна ефективність E_{loc} може бути кількісно визначена як

$$E_{loc} = \frac{1}{N}\sum_{i \in G_i} E_{glob}(G_i), \tag{14.35}$$

де G_i — локальний підграф G, а $E_{glob}(G_i)$ характеризує ефективність цього конкретного підграфа. Подібно до глобального коефіцієнта кластеризації, E_{loc} визначає, наскільки відмовостійкою є досліджувана система, тобто наскільки ефективним є транспортування інформації між першими сусідами i-го вузла при його видаленні.

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="orange")
```

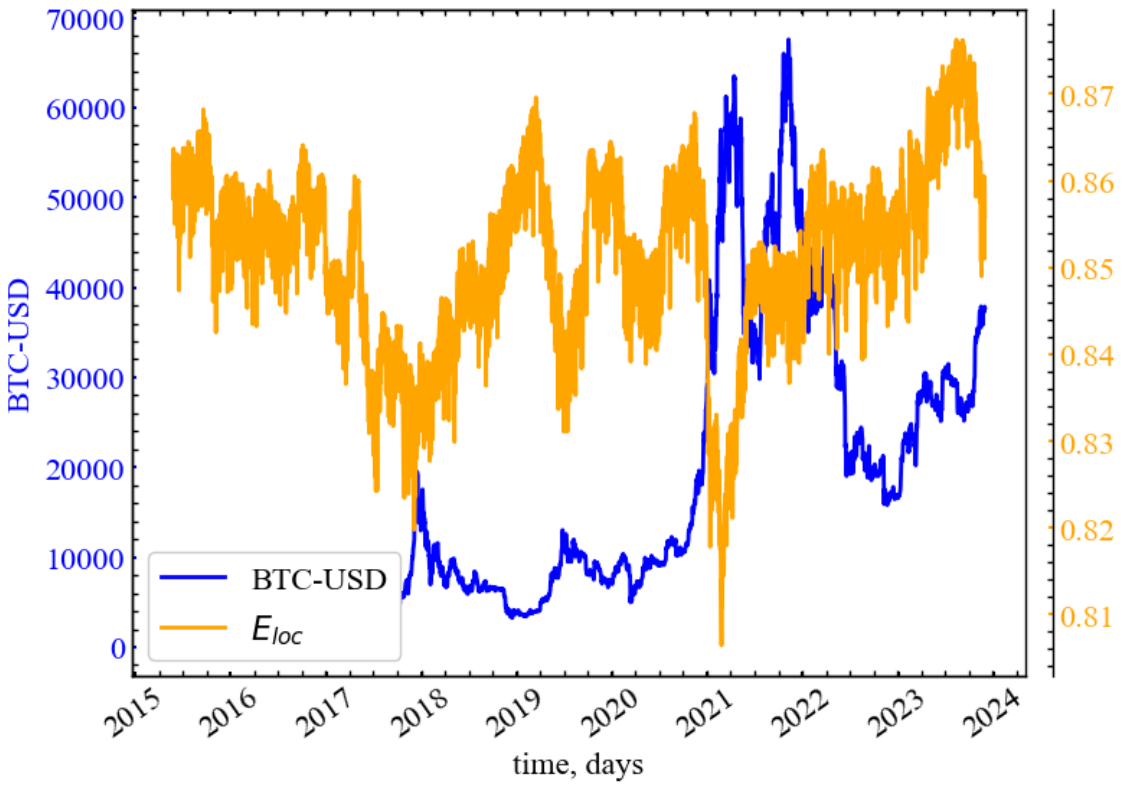


Рис. 14.24: Динаміка індексу BTC і локальної ефективності мережі

Глобально, $E_{loc} \approx 0.85$ для ринку Біткоїна, що вказує на глобальну стійкість криптовалютної мережі до можливих атак і виключень трейдерів ринку з глобальної торгівлі. Віконна процедура показує, що E_{loc} спадає в передкризові періоди, що вказує на спад локальної ефективності мережі. Як уже зазначалося, оскільки увага більшості спрямовується на одного або декількох крупних гравців ринку, їх потенційне відключення з глобальної торгівлі могло б дестабілізувати весь криптовалютний ринок.

14.2.2.8 Найкоротший шлях

```
AvgPathLength = []

for i in tqdm(range(0, length-window, tstep)):
# відбираємо фрагмент
    fragm = time_ser.iloc[i:i+window].copy()

# виконуємо процедуру трансформації ряду
    fragm = transformation(fragm, ret_type)

    if graph_type == 'classic':
        g = NaturalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()
    if graph_type == 'horizontal':
        g = HorizontalVG(directed=None).build(fragm)
        pos = g.node_positions()
        nxg = g.as_networkx()

# розрахунок середньої довжини найкоротшого шляху
    avg_path_len = nx.average_shortest_path_length(nxg)

    AvgPathLength.append(avg_path_len)

ind_names = ['AvgPathLength']

indicators = [AvgPathLength]

measure_labels = [r'$ApLen$']

file_names = []

for i in range(len(ind_names)):
    name =
f"{ind_names[i]}_symbol={symbol}_wind={window}_step={tstep}_seriestype={ret_type
}_graph_type={graph_type}"
    np.savetxt(name +".txt", indicators[i])
    file_names.append(name)
```

14.2.2.8.1 Середня довжина найкоротшого шляху

Звертаючи увагу на довжину найкоротшого шляху між двома вершинами i та j, ми можемо визначити таку міру, як **середня довжина найкоротшого шляху** (average shortest path length):

$$ApLen = \frac{1}{N(N-1)} \sum_{i \neq j} l_{ij}. \tag{14.36}$$

```
plot_pair(time_ser.index[window:length:tstep],
          time_ser.values[window:length:tstep],
          indicators[0],
          ylabel,
          measure_labels[0],
          xlabel,
          file_names[0],
          clr="deeppink")
```

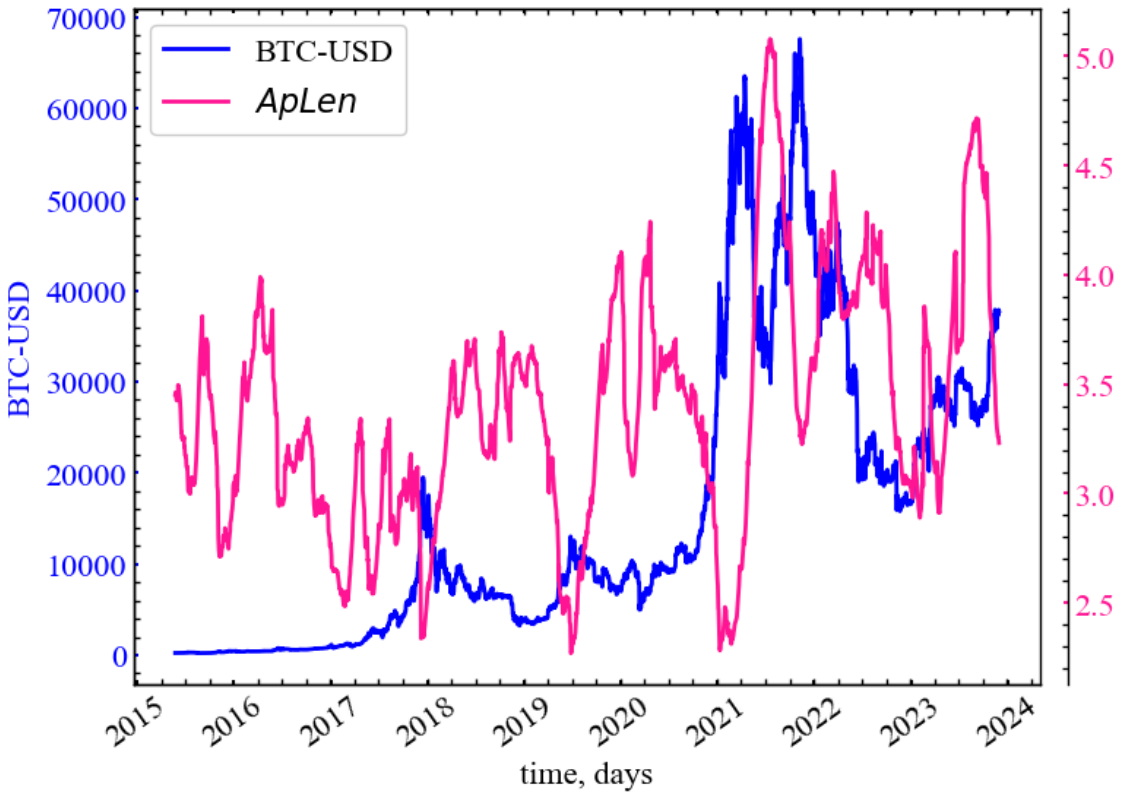


Рис. 14.25: Динаміка індексу BTC і середньої довжини найкоротшого шляху

На Рис. 14.25 продемонстровано, що $ApLen$ характеризується спадом у передкризові періоди та зростанням у кризові й посткризові періоди. Подібно до попередніх індикаторів, що тільки опиралися на довжини найкоротшого шляху між парами вершин, $ApLen$ вказує на зростання ефективності передачі інформації між трейдерами ринку. Також можна сказати, що на побудованому

криптовалютному графі видимості минулі значення “бачать” теперішні в більш ефективний спосіб, що відображається в персистентності ринку в передкризовий період.

14.3 Висновок

У даній роботі було продемонстровано можливість дослідження складних соціально-економічних систем в рамках мережної парадигми складності. Часовий ряд Біткоїна був представлений еквівалентним чином — мережею видимості, яка має широкий спектр характеристик: і спектральних, і топологічних. Приклади криптовалютних крахів показали, що більшість мережних показників можуть слугувати індикаторами-передвісниками кризових явищ і можуть бути використані для можливого раннього попередження небажаних криз на криптовалютних ринках.

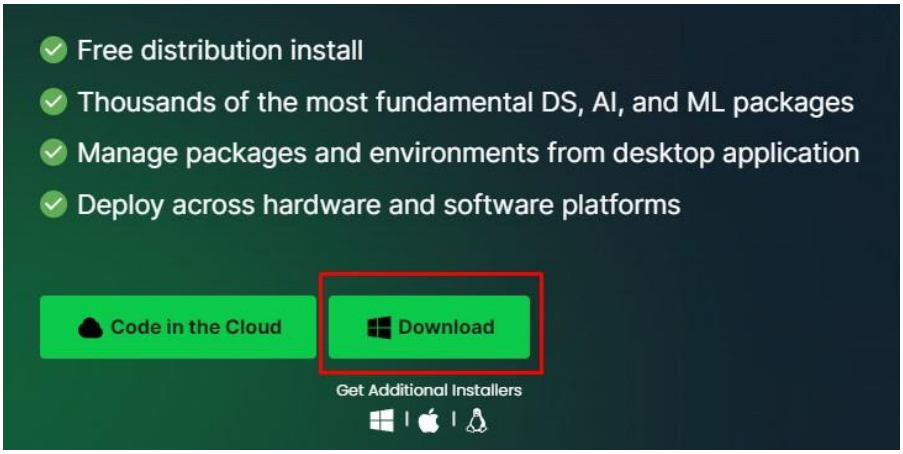
14.4 Завдання для самостійної роботи

1. Для заданих часових рядів чи їх сукупності побудувати всі види мереж, дослідити їх графодинаміку, порівняти результати і зробити висновки щодо їх прогностичних можливостей
2. Проаналізувати результати як для вихідного ряду, так і для стандартизованих прибутковостей.
3. Як змінюються результати для не фінансових часових рядів. Чи спостерігається універсальність результатів?

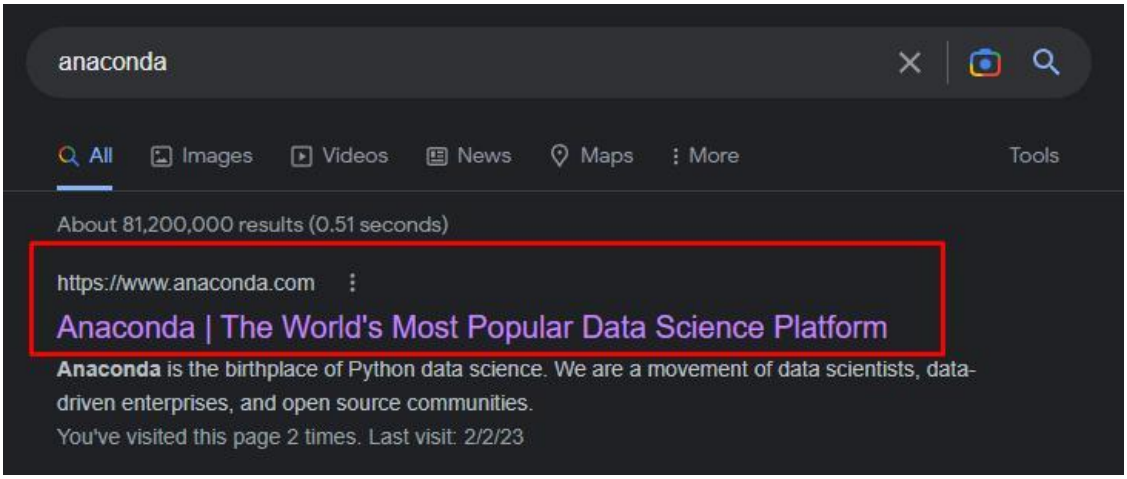
Appendix A — Інструкція зі встановлення Anaconda Navigator

1. Відвідайте сторінку Anaconda за наступним посиланням: https://www.anaconda.com/download/. Ви маєте побачити наступне зображення:

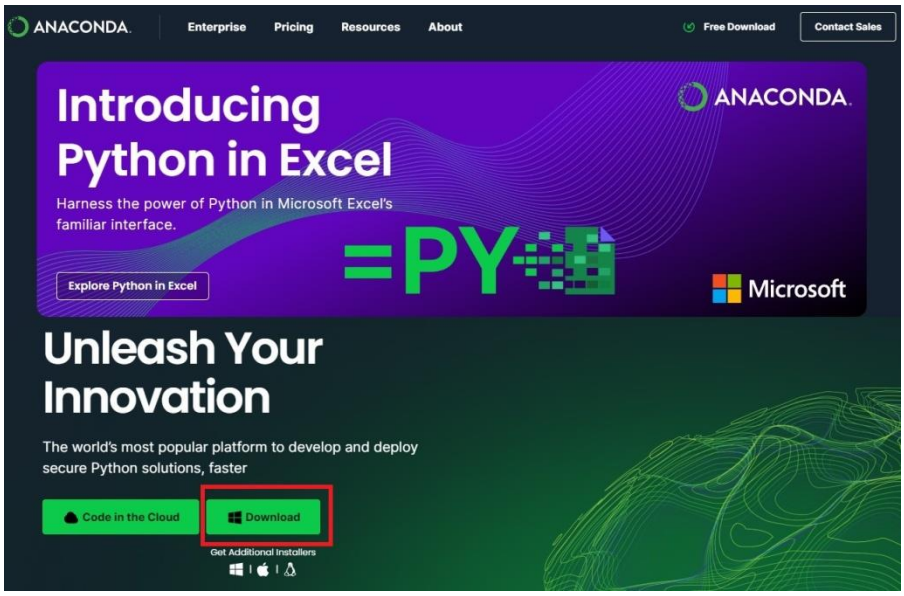
2. Натискаємо на кнопку **Download**:



3. Якщо ви набиратимете **anaconda** у пошуковому рядку, тоді потрібно перейти за наступним посиланням, що має з’явитися найпершим:

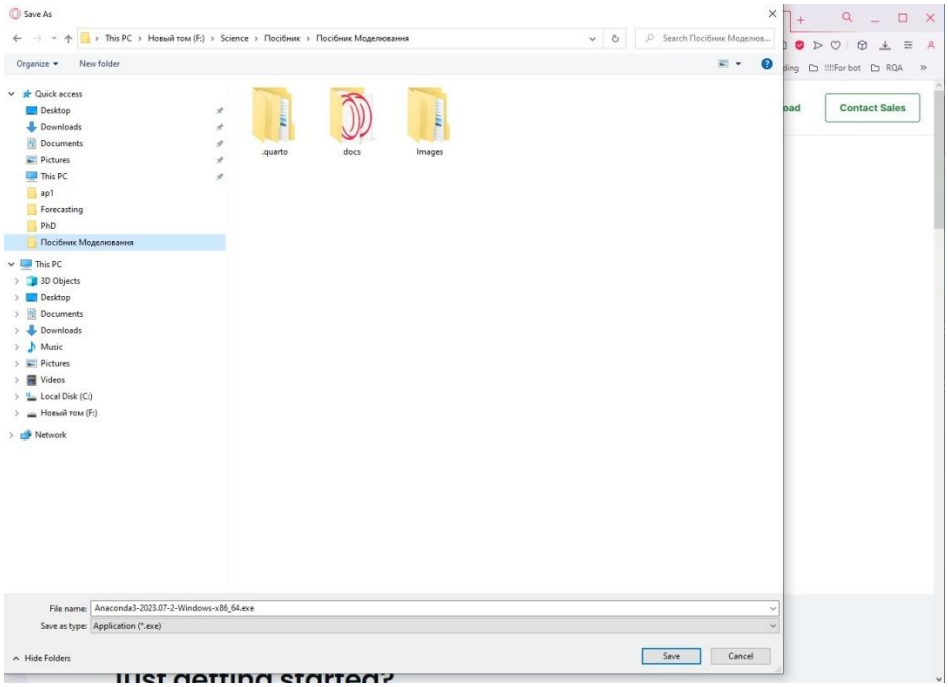


Перейшовши по виділеному посиланню, маєте побачити наступну сторінку:

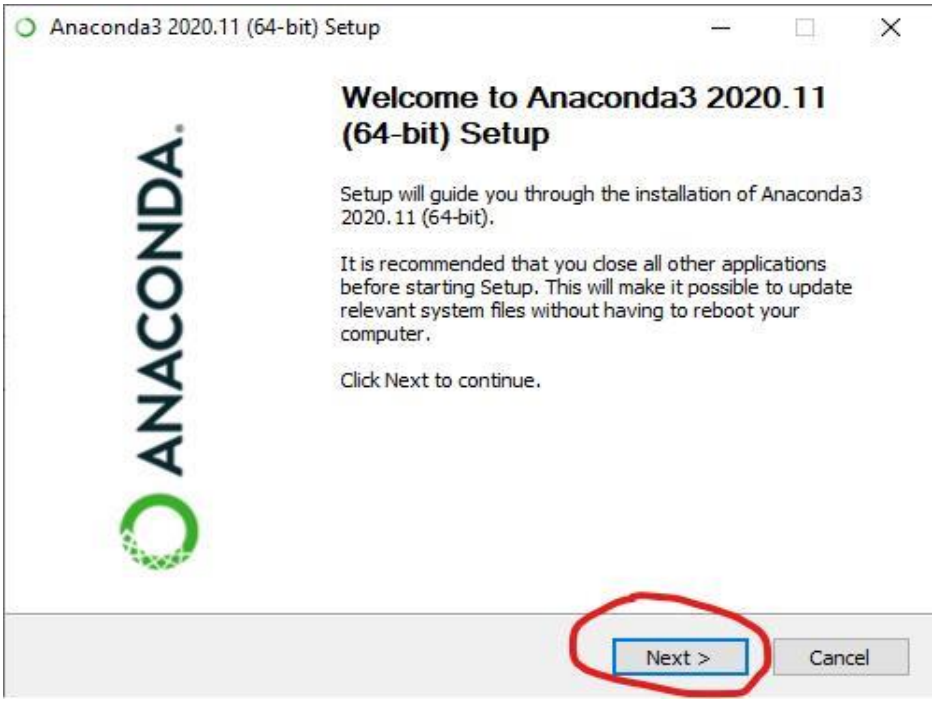


Натиснемо на кнопку **Download**, щоб розпочати встановлення.

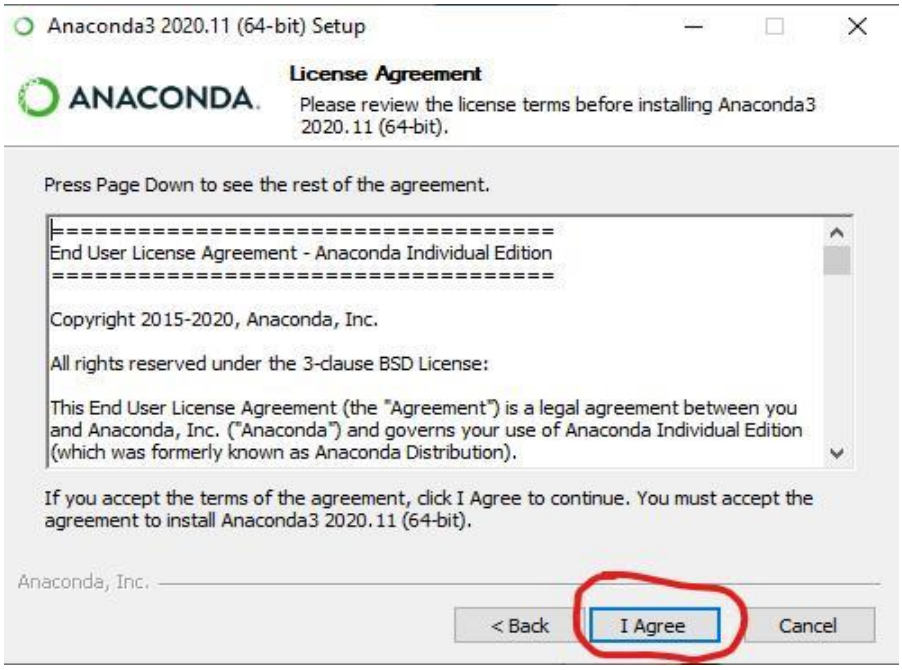
4. З’явиться вікно наступного виду і запропонує зберегти файл там, де це потрібно:



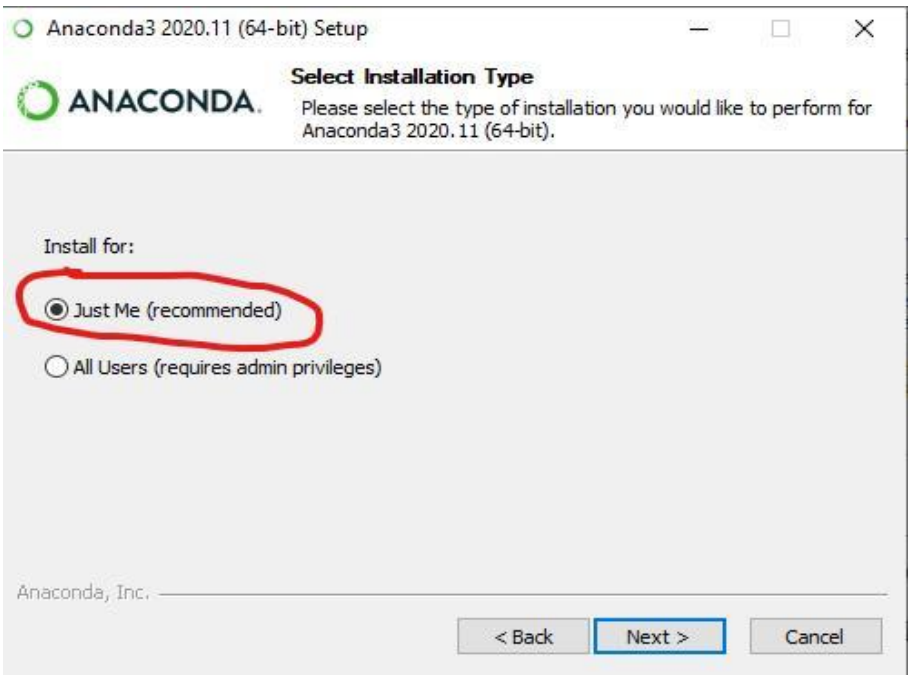
5. Натискаємо на клавішу **Next**:



6. Уважно читаємо ліцензійну угоду та натискаємо клавішу **I Agree**:

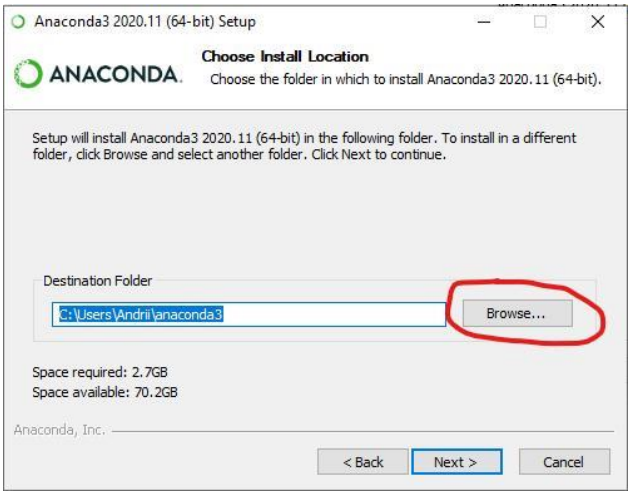


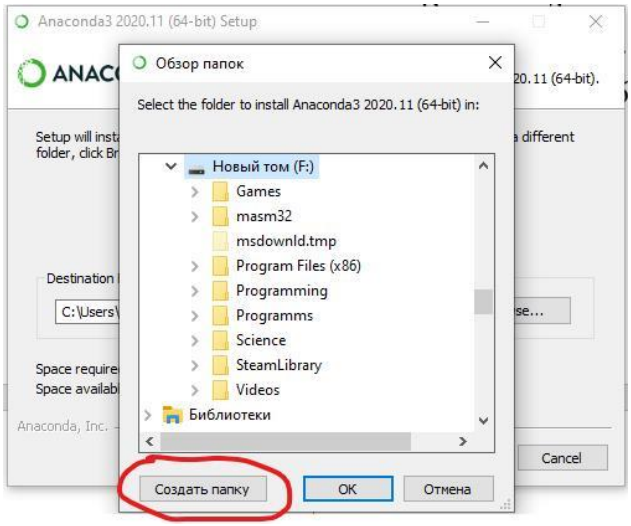
7. Далі, якщо ви не перебуваєте у команді розробників, і ви єдина людина, хто буде користуватися Anaconda — **Just me** ваш вибір:

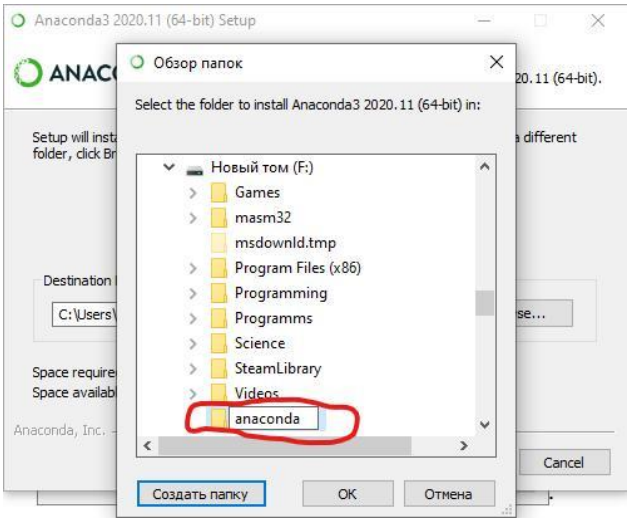


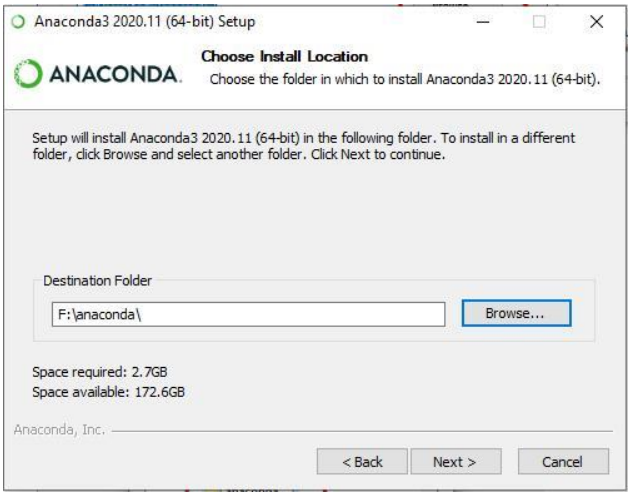
Натискаємо **Next**.

8. На наступному кроці 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 **Створити папку**. Таким чином ми створимо в нашому диску папку з назвою anaconda до якої і буде завантажено Anaconda.







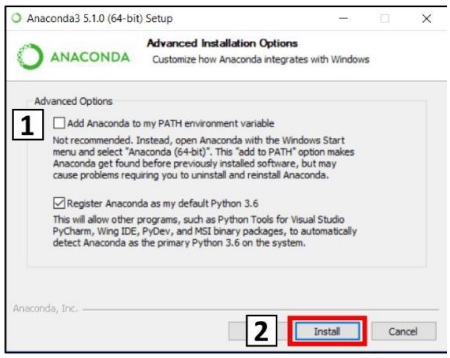


9. Наступним кроком буде обрання середовища змінних (the environment variables).

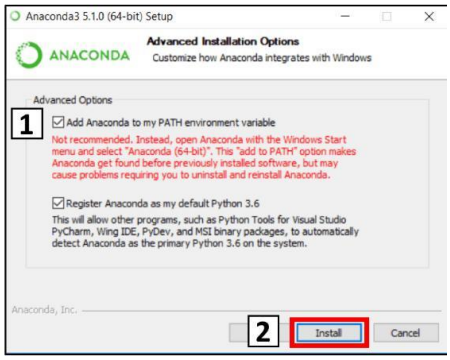
   Якщо ви встановлюєте Python вперше, відмітимо **Add Anaconda to my PATH environment variable**. Це дасть вам можливість використовувати Anaconda в командному рядку (або Git bash, cmder, powershell і т.д.).

   Якщо ви вже маєте Python на своєму комп'ютері, тоді прапорець не варто відмічати. Ви зможете запускати Anaconda Navigator або Anaconda Command Prompt (розташовані в меню **Пуск** у розділі Anaconda), якщо 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.

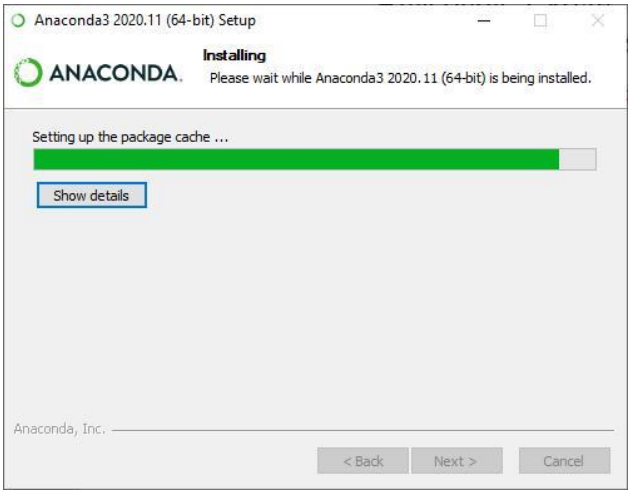


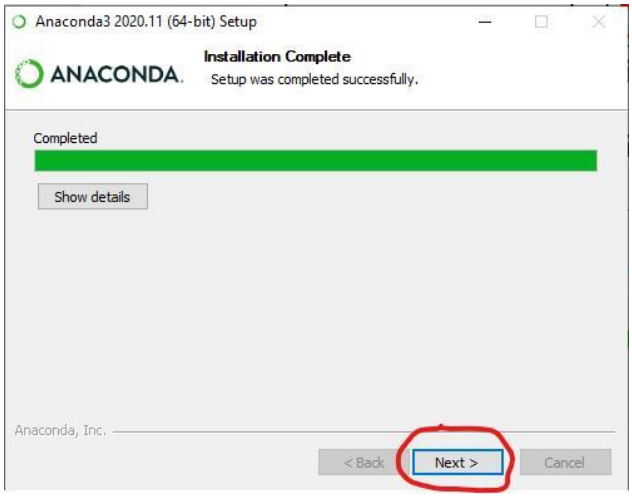
Рекомендований підхід



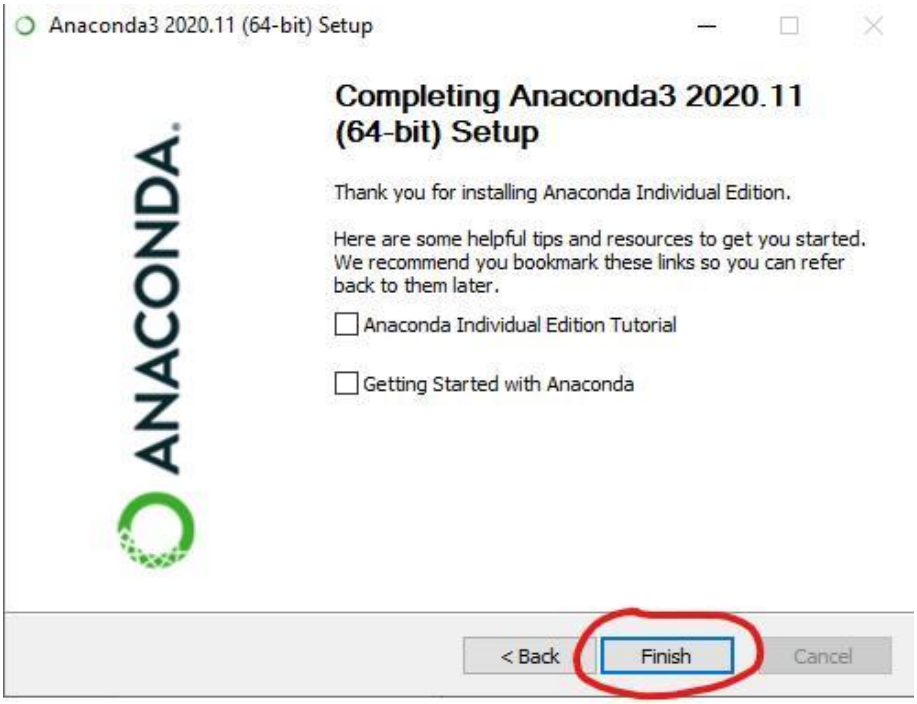
Альтернативний

10. Натискаємо **Install** і чекаємо завершення:

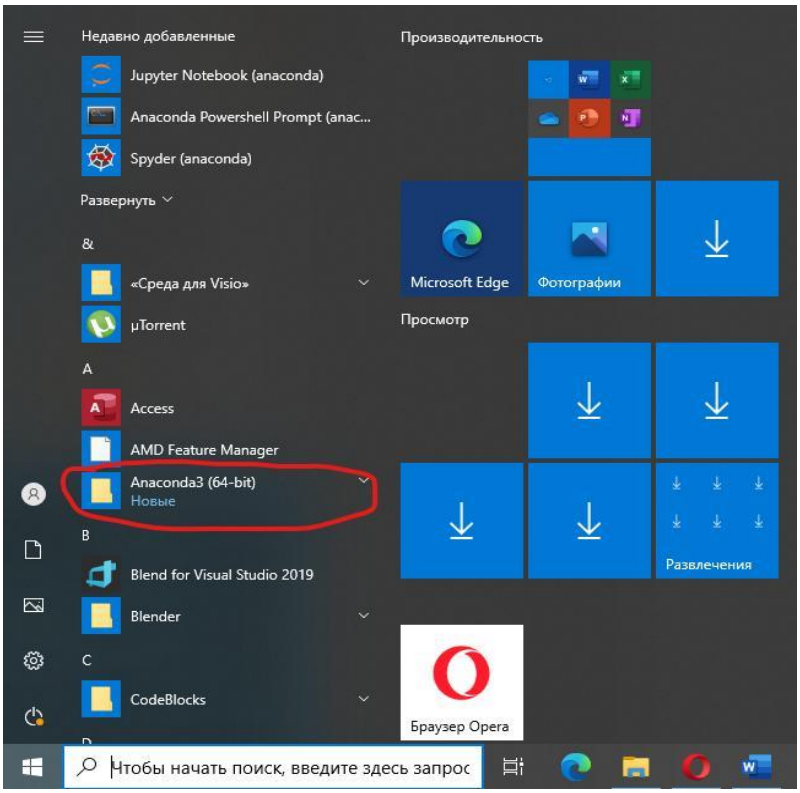




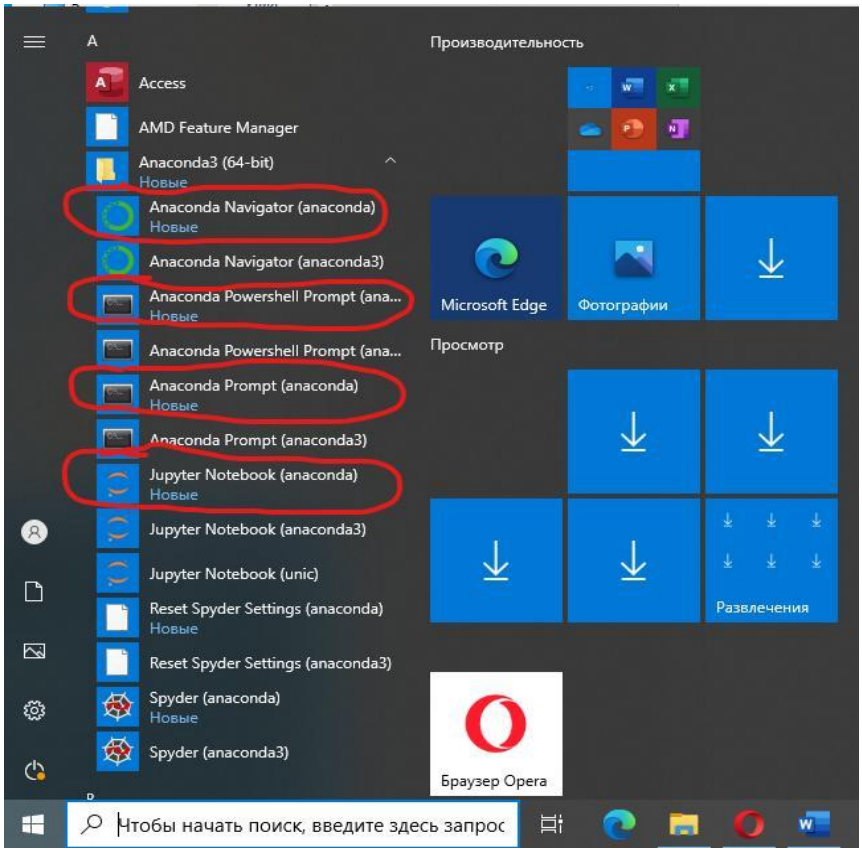
11. Натискаємо **Next** аж до вікна з подякою за встановлення Anaconda і після натискаємо **Finish**:



12. На наступному кроці потрібно перейти в меню **Пуск** і знайти папку під назвою Anaconda:



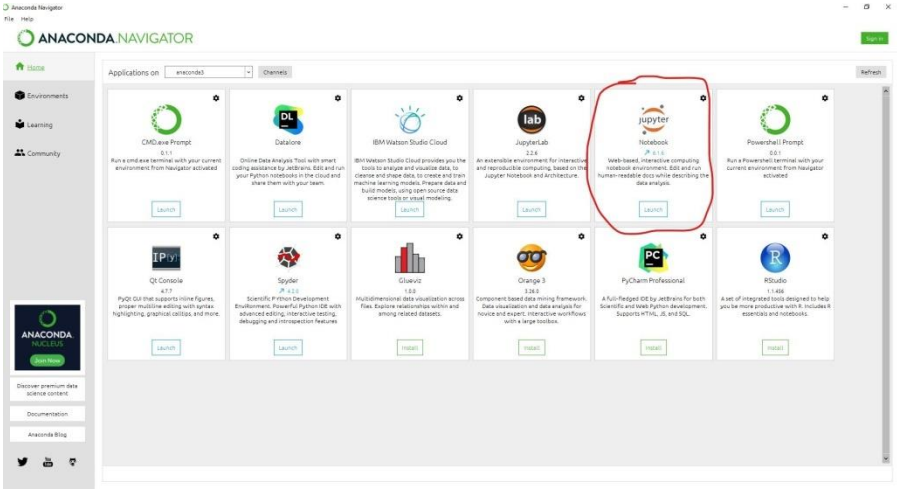
13. Розгорнувши папку можна побачити цікаві для нас іконки:



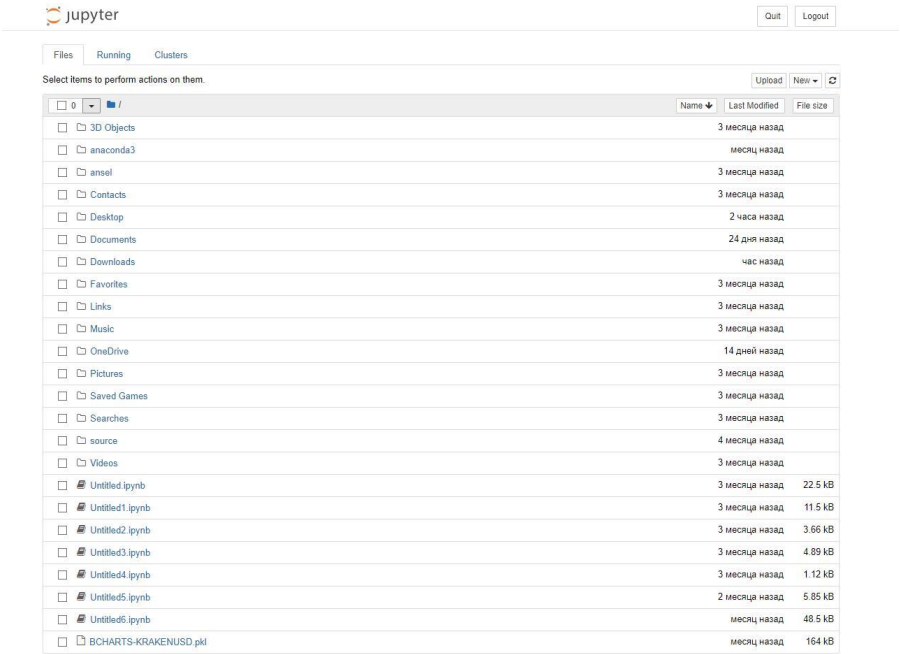
На зазначеній вище фотографії можна бачити декілька ярликів Anaconda Navigator, Anaconda Powershell Promt, Anaconda Promt та Jupyter Notebook з

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.

14. Натиснувши на Anaconda Navigator, потрапляємо до середовища, що пропонує різноманітні інструменти для аналізу даних. Для подальшої роботи нам знадобиться лише Jupyter Notebook:

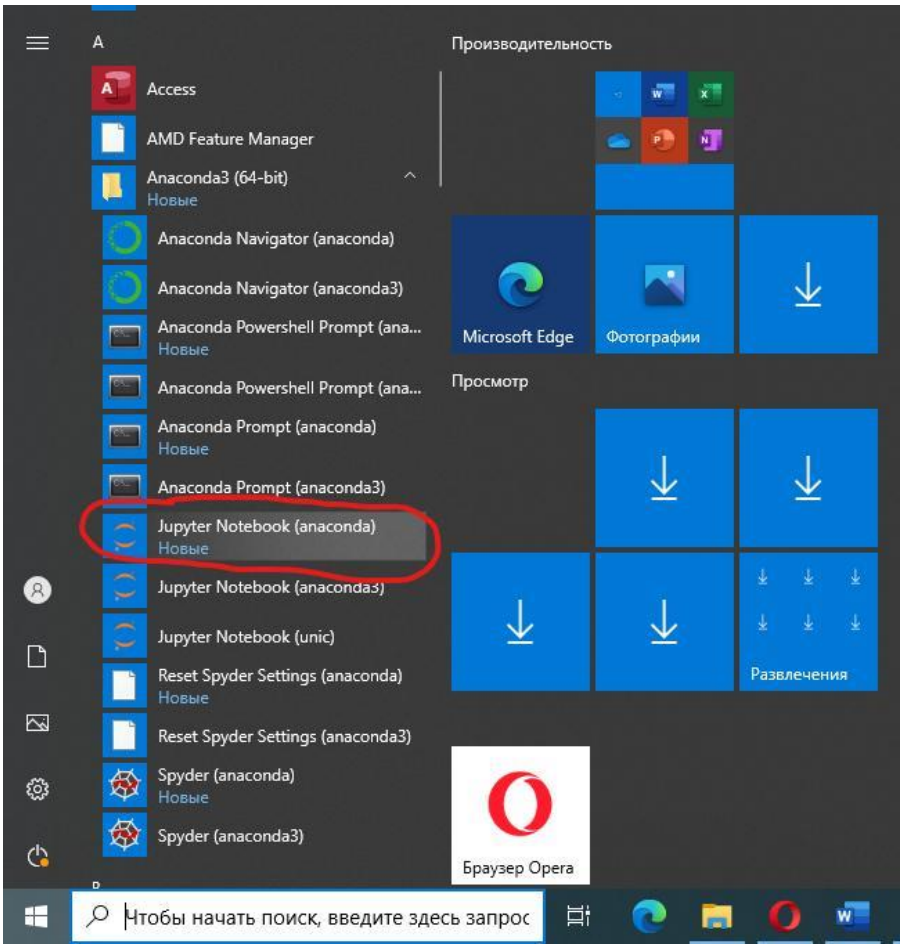


Натискаємо на **Launch** та потрапляємо до середовища Jupyter (кореневої папки до якої було встановлено anaconda):

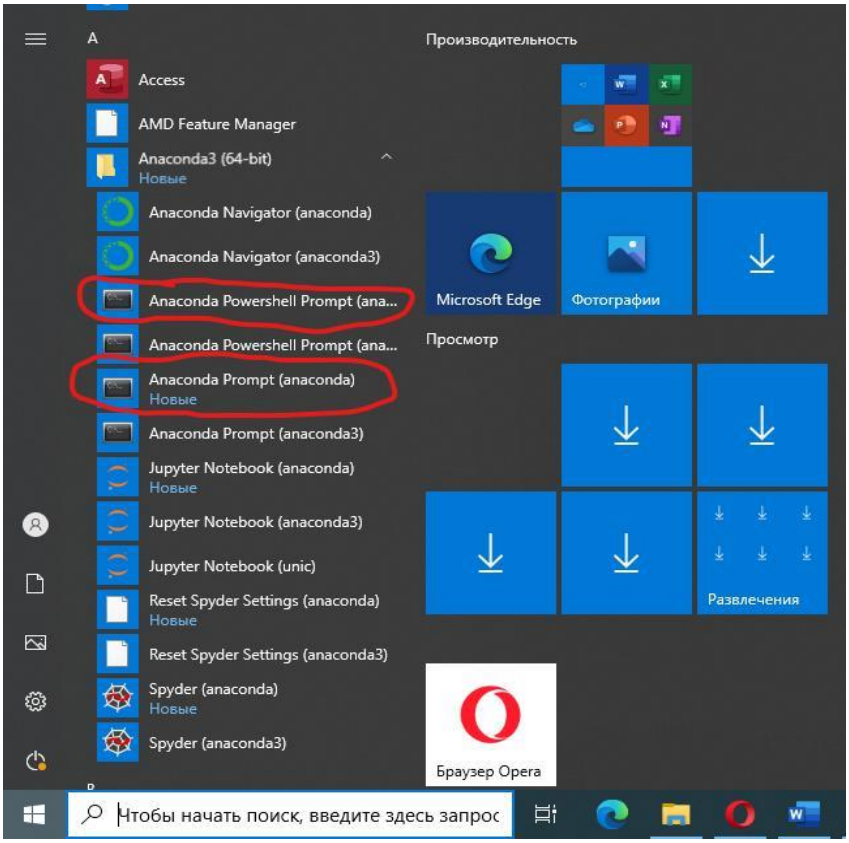


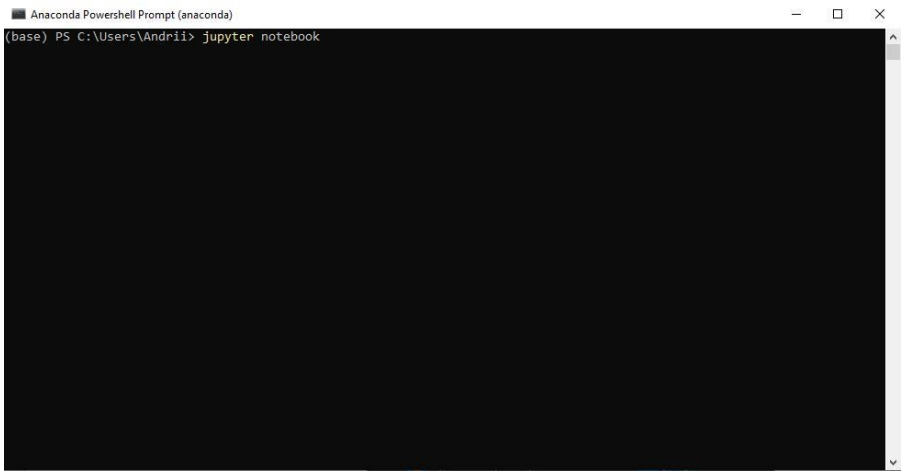
У верхньому лівому кутку буде знаходитись значок **New**. Спочатку натискаємо на нього і потім на **Python 3**. Має створитися відповідний Notebook у якому можна писати код:

15. Окрім цього, в меню пуск можна натиснути на значок Jupyter Notebook та одразу перейти до роботи:



16. 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`**:


A
Access
AMD Feature Manager
Anaconda3 (64-bit)
Новые
Anaconda Navigator (anaconda)
Anaconda Navigator (anaconda3)
Anaconda Powershell Prompt (ana...
Anaconda Powershell Prompt (ana...
Anaconda Prompt (anaconda)
Новые
Anaconda Prompt (anaconda3)
Jupyter Notebook (anaconda)
Новые
Jupyter Notebook (anaconda3)
Jupyter Notebook (unic)
Reset Spyder Settings (anaconda)
Новые
Reset Spyder Settings (anaconda3)
Spyder (anaconda)
Новые
Spyder (anaconda3)
Производительность
Microsoft Edge
Фотографии
Просмотр
Развлечения
Браузер Opera
Чтобы начать поиск, введите здесь запрос


Anaconda Powershell Prompt (anaconda)
(base) PS C:\Users\Andrii> 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 відстеження того, що відбувається. Все, що міститься в коментарі, зазвичай ігнорується при фактичному запуску коду, але 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 решітки (`# 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`, що називається `docstring` і використовується для пояснення призначення функції.

```
""" This is a special string """
```

```
' This is a special string '
```

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 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, 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 знака рівності (=). Змінні чутливі до регістру, тому будь-які зміни в заголовних літерах імені змінної будуть посилатися на іншу змінну.

```
one = 1
print(one)

1
```

Число з плаваючою комою або `float` — це 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 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!

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 15%4, він поверне залишок після ділення 15 на 4.

```
print('Частка: ', 15 % 4)
```

```
Частка:  3
```

Математичні функції також працюють зі змінними!

```
first_integer = 4
second_integer = 5
print(first_integer * second_integer)
```

```
20
```

Якщо ви виконуєте математику виключно з цілими числами, ви отримуєте ціле число. Включення будь-якого значення з плаваючою точкою в обчислення зробить уже результат із плаваючою точкою.

```
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 0). 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 ній ви знайдете 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 n, перший елемент списку знаходиться з індексом 0, другий елемент з індексом 1, і так далі, і тому подібне. Останній елемент списку матиме індекс $n-1$. 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, ви можете змінювати елементи списку без зміни самого списку. Деякі типи даних, такі як рядки, є незмінними. 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.

```
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 (‘.’).

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 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()`документацію 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.

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 1 або 0). 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, P і Q. 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.

```
# Схоже 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 `elif`-сів, 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 i має значення 2')
elif i == 3:
   print('Змінна i має значення 3')
else:
   print("Мене не хвилює змінна i")
```

```
Мене не хвилює змінна i
```

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('i ділиться як на 2, так і на 3!')
   elif i % 5 == 0:
      print('i 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.')

i 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.

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.

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 до виконання певної умови. Якщо в будь-який момент після ітерації ця умова більше не виконується, цикл завершується. Цикл `for` буде виконувати ітерацію за послідовністю значень і завершиться, коли послідовність закінчиться. Можливо включити умови в цикл `for`, щоб вирішити, чи, можливо, 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! 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, наприклад, при вказівці кільності ітерацій в ітераторі `range`. У цьому випадку ми проходимось по списку, що був 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 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 3 через інструкцію `continue`, яка виконується, коли ми вводимо оператор if. Код ніколи не бачить команди для друку числа 3, 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 повертала (`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 ввести додаткові параметри, розділивши їх комами.

```
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
```

Ми можемо визначити функцію так, щоб вона приймала довільну кількість параметрів. 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
```

Використовуйте `*args` як параметр для вашої функції, 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, що 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, що 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 заглибитися в матеріал, 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. 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, $e^{i\pi} = -1$) або створити в окремому рядку:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

Для редагування тексту в markdown осередку необхідно два рази клікнути по ній.

C.2 Комірка Code

Наступна комірка є Code 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([0.66066316, 0.72739371, 0.52981988, 0.49703974, 0.48385109,
       0.12894506, 0.13740195, 0.81722274, 0.05088196, 0.9682992 ])
```

С.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?

Type:        module
String form: <module 'numpy' from 'c:\\Users\\Andrii\\anaconda3\\Lib\\site-
packages\\numpy\\__init__.py'>
File:        c:\users\andrii\anaconda3\lib\site-packages\numpy\__init__.py
Docstring:
NumPy
=====

Provides
  1. An array object of arbitrary homogeneous items
  2. Fast mathematical operations over arrays
  3. Linear Algebra, Fourier Transforms, Random Number Generation

How to use the documentation
----------------------------
Documentation is available in two forms: docstrings provided
with the code, and a loose standing reference guide, available from
`the NumPy homepage <https://numpy.org>`_.

We recommend exploring the docstrings using
`IPython <https://ipython.org>`_, an advanced Python shell with
TAB-completion and introspection capabilities.  See below for further
instructions.

The docstring examples assume that `numpy` has been imported as ``np``::

>>> import numpy as np

Code snippets are indicated by three greater-than signs::

>>> x = 42
>>> x = x + 1

Use the built-in ``help`` function to view a function's docstring::

>>> help(np.sort)
  ... # doctest: +SKIP

For some objects, ``np.info(obj)`` may provide additional help.  This is
```

```
particularly true if you see the line "Help on ufunc object:" at the top
of the help() page.  Ufuncs are implemented in C, not Python, for speed.
The native Python help() does not know how to view their help, but our
np.info() function does.

To search for documents containing a keyword, do::

>>> np.lookfor('keyword')
  ... # doctest: +SKIP

General-purpose documents like a glossary and help on the basic concepts
of numpy are available under the ``doc`` sub-module::

>>> from numpy import doc
>>> help(doc)
  ... # doctest: +SKIP

Available subpackages
---------------------
lib
    Basic functions used by several sub-packages.
random
    Core Random Tools
linalg
    Core Linear Algebra Tools
fft
    Core FFT routines
polynomial
    Polynomial tools
testing
    NumPy testing tools
distutils
    Enhancements to distutils with support for
    Fortran compilers support and more  (for Python <= 3.11).

Utilities
---------
test
    Run numpy unittests
show_config
    Show numpy build configuration
matlib
    Make everything matrices.
__version__
    NumPy version string

Viewing documentation using IPython
-----------------------------------

Start IPython and import `numpy` usually under the alias ``np``: `import
numpy as np`.  Then, directly past or use the ``%cpaste`` magic to paste
examples into the shell.  To see which functions are available in `numpy`,
type ``np.<TAB>`` (where ``<TAB>`` refers to the TAB key), or use
``np.*cos*?<ENTER>`` (where ``<ENTER>`` refers to the ENTER key) to narrow
down the list.  To view the docstring for a function, use
```

```
``np.cos?<ENTER>`` (to view the docstring) and ``np.cos??<ENTER>`` (to view
the source code).

Copies vs. in-place operation
-----------------------------
Most of the functions in `numpy` return a copy of the array argument
(e.g., `np.sort`).  In-place versions of these functions are often
available as array methods, i.e. ``x = np.array([1,2,3]); x.sort()``.
Exceptions to this rule are documented.
```

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)]
```

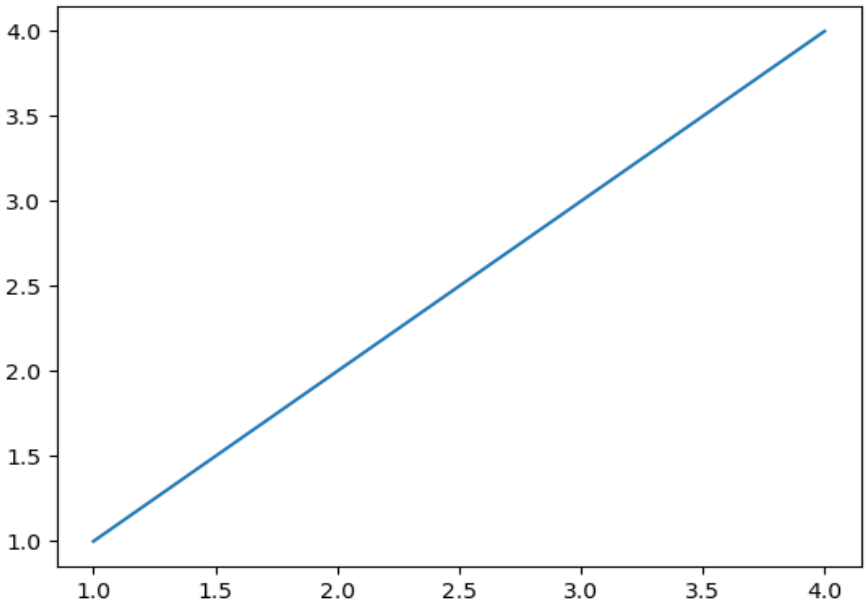
```
377 µs ± 21.4 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Список усіх magic команд можна подивитися окремою командою `%lsmagic`.

C.5 Робота з графікою

У Python є багато бібліотек для візуалізації даних. Більшість з них інтегруються з Jupyter і відображають графіки.

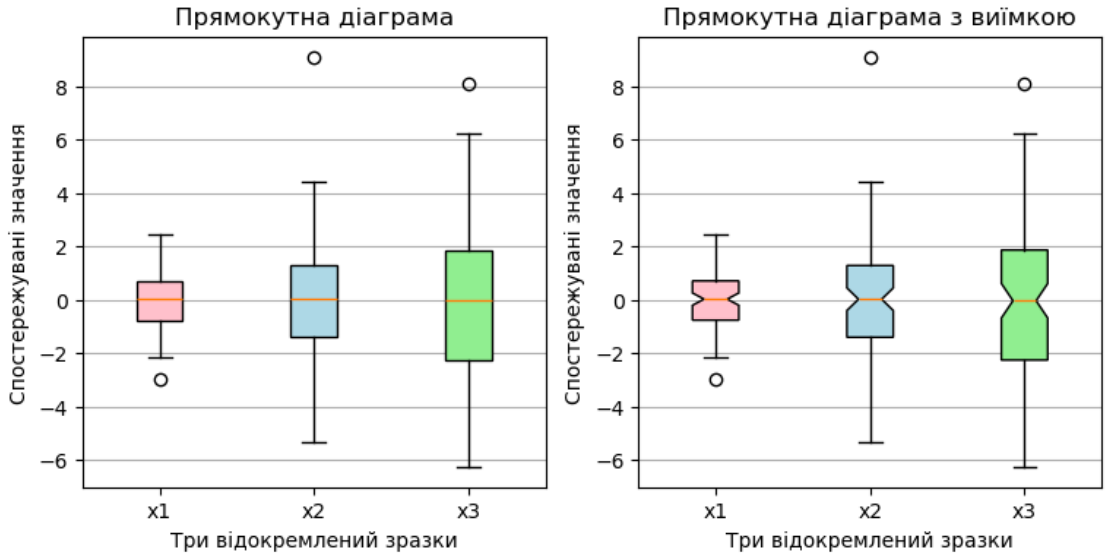
```
import matplotlib.pyplot as plt # імпортуємо бібліотеку

# вбудовуємо виведені рисунки в юпітеровський ноутбук
%matplotlib inline

plt.plot([1, 4], [1, 4]);# виводимо лінію за двома точками
```



Або будуємо декілька графіків:

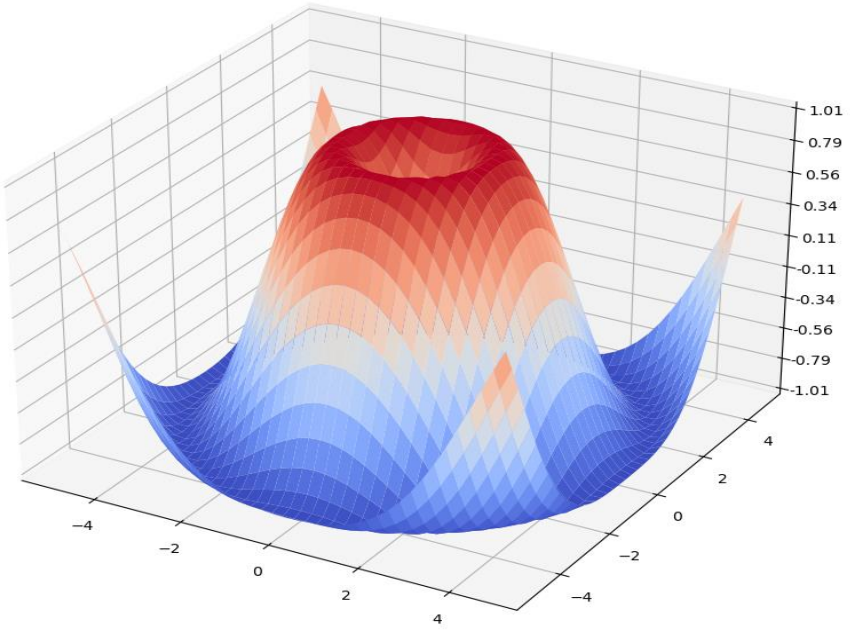
```
all_data = [np.random.normal(0, std, size=100) for std in range(1, 4)] #
генеруємо список значень із нормального розподілу
labels = ['x1', 'x2', 'x3'] # ініціалізуємо список міток

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 на вісі 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 на вісі 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 inzip(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:

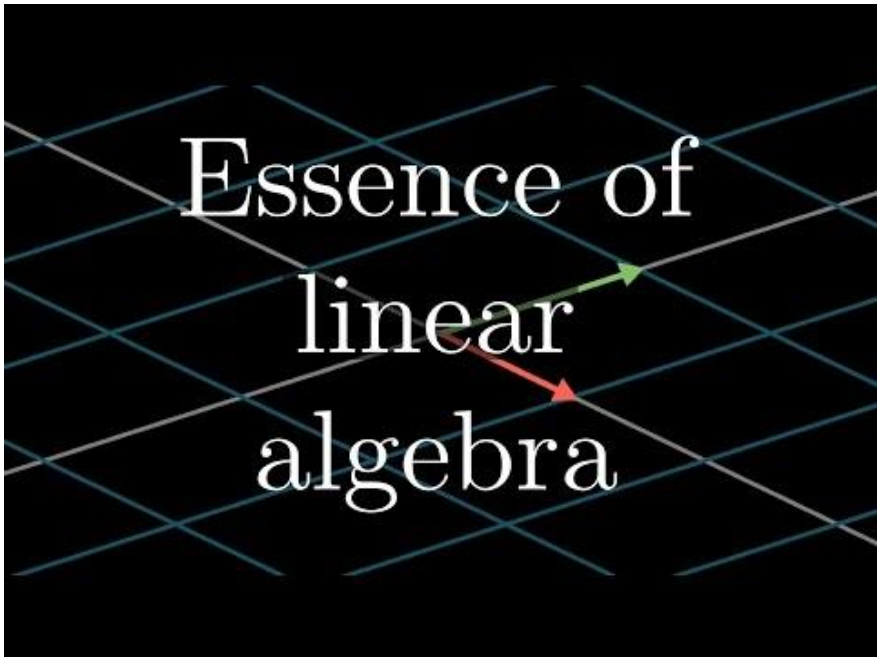
```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np

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 за двома вісями
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

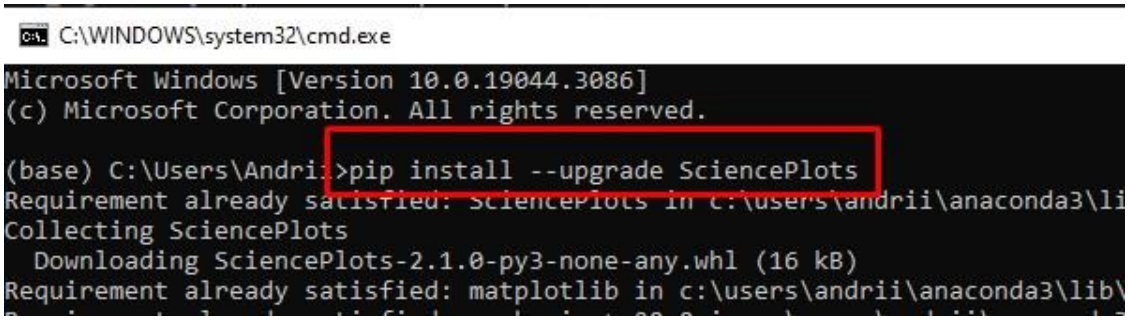
Для Jupyter Notebook було створено велику кількість плагінів. Наприклад, можна вбудовувати відео з youtube:

```
from IPython.display import YouTubeVideo
YouTubeVideo('kjBOesZCoqc')
```



Або інтерактивні карти (дана комірка відобразиться тільки якщо у вас встановлений folium. 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

import folium
m = folium.Map(zoom_start=12, location=[47.89829743895897, 33.36626740165739])
m

<folium.folium.Map at 0x25c9502a150>
```

Або вбудувати будь-який інший шматок HTML за допомогою магічної 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%5Etf
w">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 так званих гарячих клавіш. Список гарячих клавіш можна знайти в меню *Help – Keyboard shortcuts*. Нижче наведено список найбільш корисних поєднань:

Ключ	Опис
Esc	вийти з режиму редагування та виділити поточну комірку
Enter	перейти в режим редагування 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

Ключ	Опис
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

Appendix D — Вступ до Google Colab

D.1 Перші кроки роботи в Google Colab

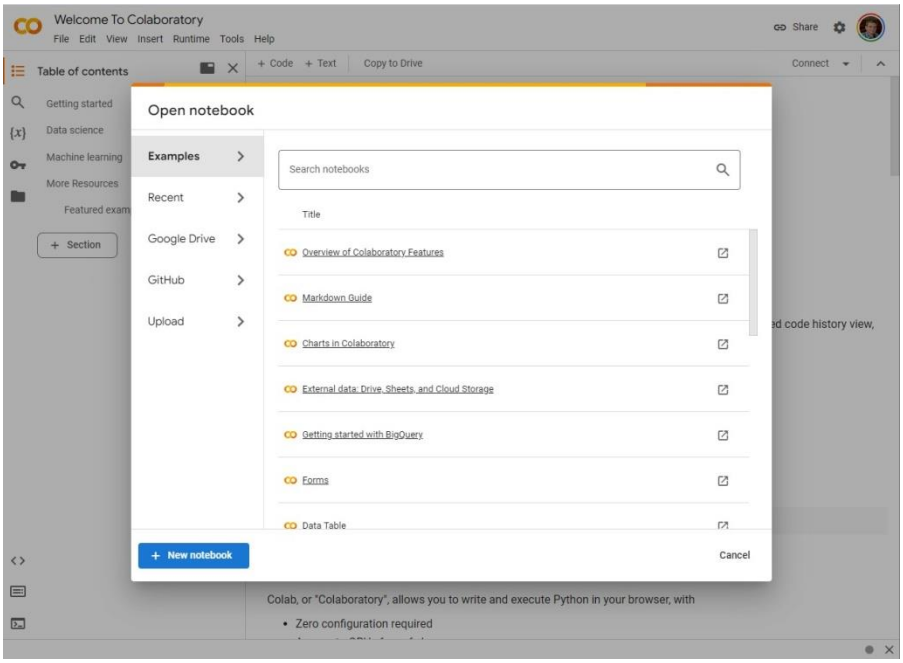
Google Colab (Collaboratory) — це потужна хмарна платформа, яка дозволяє писати і виконувати код на Python через веб-браузер. Вона надає можливість працювати у середовищі Jupyter Notebook без жодних налаштувань чи інсталяції на вашому локальному комп'ютері. Colab особливо корисний для задач машинного навчання, аналізу даних та наукових обчислень, оскільки пропонує безкоштовний доступ до графічних процесорів та попередньо встановлених бібліотек Python, таких як TensorFlow, Keras, PyTorch, NumPy, Pandas, Matplotlib та Scikit-learn.

Як програміст, ви можете виконувати наступні дії за допомогою Google Colab:

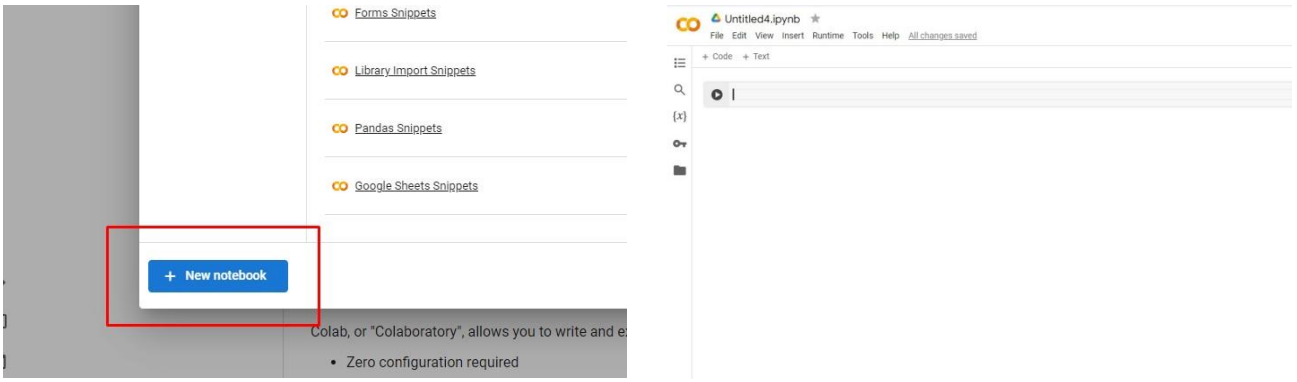
- писати та виконувати код на Python;
- документувати свій код;
- створювати/завантажувати/поширювати Jupyter блокноти;
- імпортувати/зберігати блокноти з/на Google Диск;
- імпортувати/публікувати блокноти з GitHub;
- імпортувати зовнішні набори даних, наприклад, з Kaggle;
- інтегрувати PyTorch, TensorFlow, Keras, OpenCV;
- мати безкоштовний хмарний сервіс з графічним процесором.

> ⓘ Примітка
>
> Оскільки Colab неявно використовує Google диск для зберігання ваших блокнотів, перш ніж продовжувати роботу, переконайтеся, що ви увійти до свого облікового запису Google

Відкрийте у своєму браузері URL-адресу: https://colab.research.google.com. У вашому браузері відобразиться наступна сторінка (за умови, що ви увійшли до свого Google акаунта):

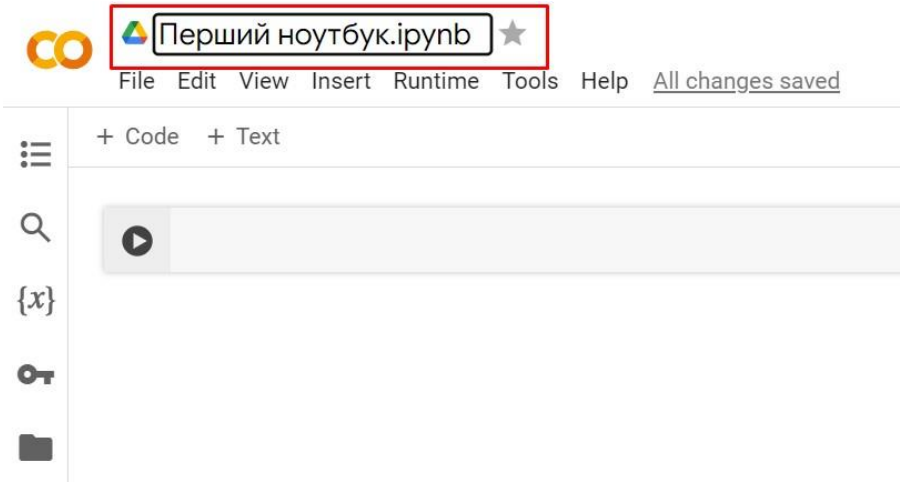


У нижньому лівому куті натисніть на кнопку `+ New notebook`, з'явиться робочий блокнот:



Як ви могли помітити, інтерфейс Colab дуже схожий на інтерфейс Jupyter. Тут є вікно коду, в якому ви можете вводити свій Python код.

За замовчуванням Colab блокнот використовує іменування `Untitled.ipynb`. Щоб перейменувати блокнот, клацніть на цій назві і введіть бажану назву у вікні редагування, як показано нижче:

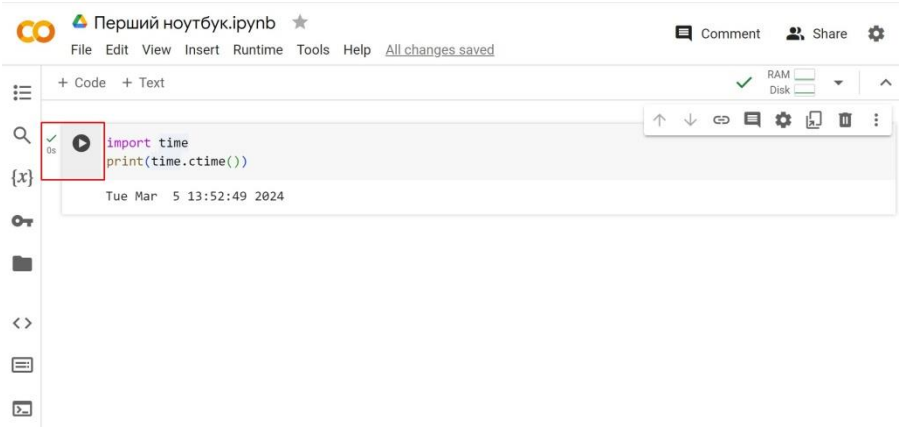


Цей блокнот буде названо `Перший ноутбук`. Отже, введіть цю назву у вікні редагування і натисніть клавішу **ENTER**. Блокнот встановить введене ім’я.

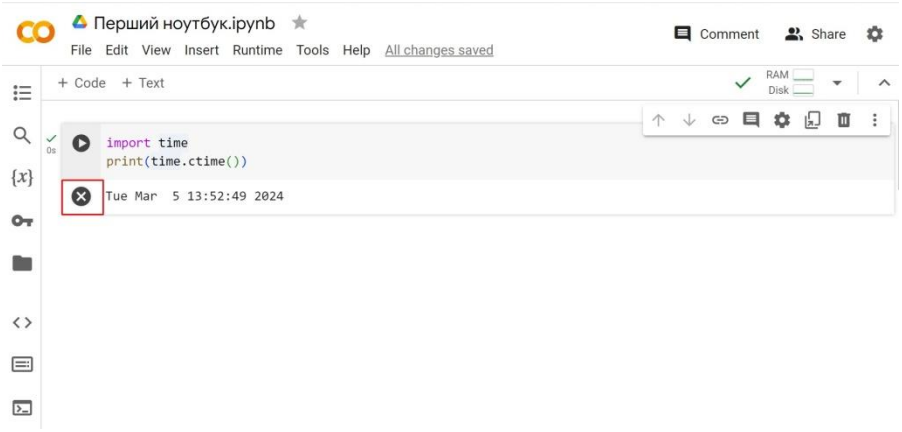
Тепер введемо тривіальний код на Python у вікні коду:

```
import time
print(time.ctime())
```

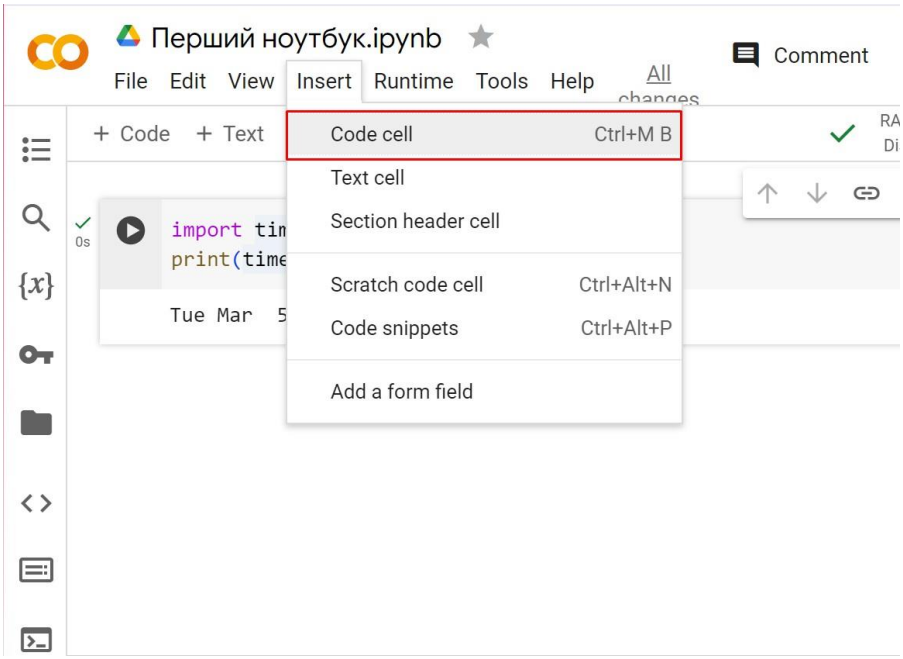
Щоб виконати код, натисніть на стрілку в лівій частині комірки коду:



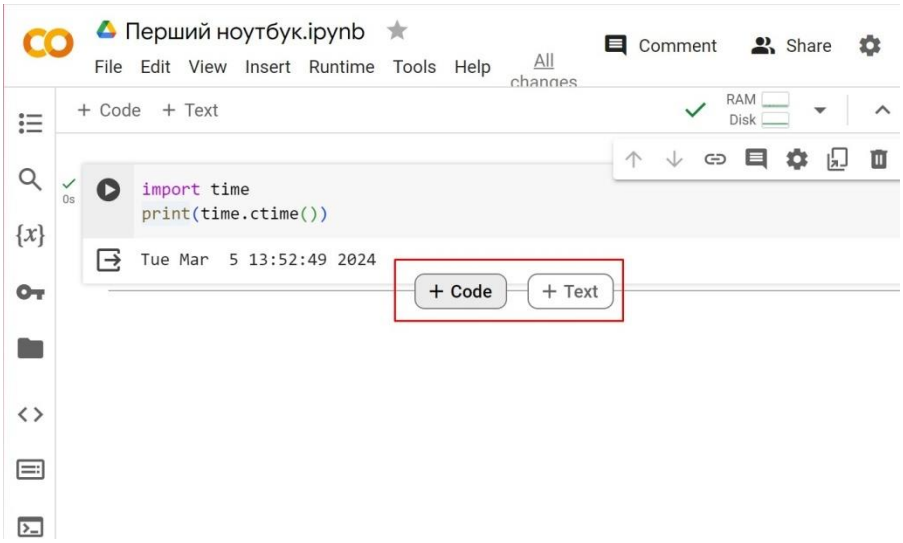
Результатом виконання є `Tue Mar 5 13:52:49 2024`. Можна будь-коли очистити вихідні дані, натиснувши на іконку зліва від дисплея вихідних даних:



Щоб додати більше комірок для написання коду, виберіть наступні пункти меню:



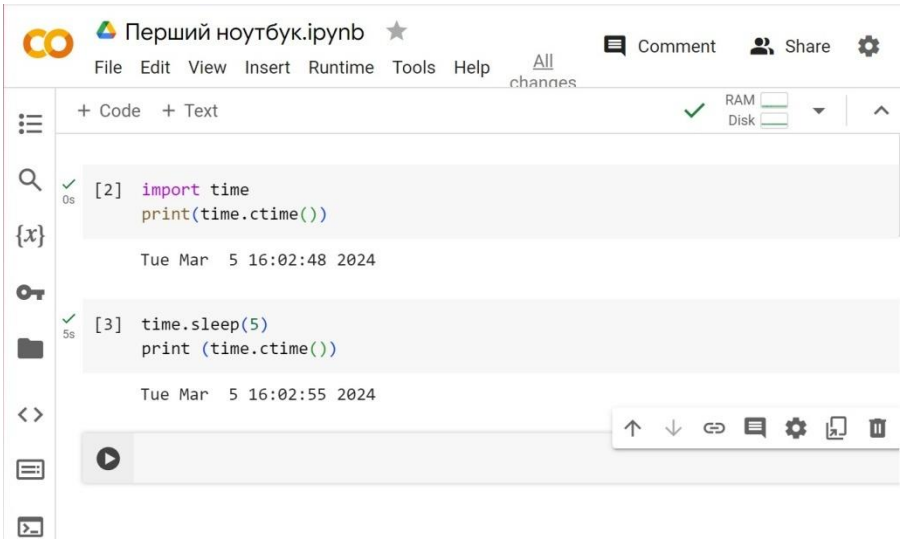
У якості альтернативи можна навести вказівник миші на нижню центральну частину клітинки коду. Коли з'являться кнопки **CODE** і **TEXT**, натисніть на **CODE**, щоб додати нову комірку:



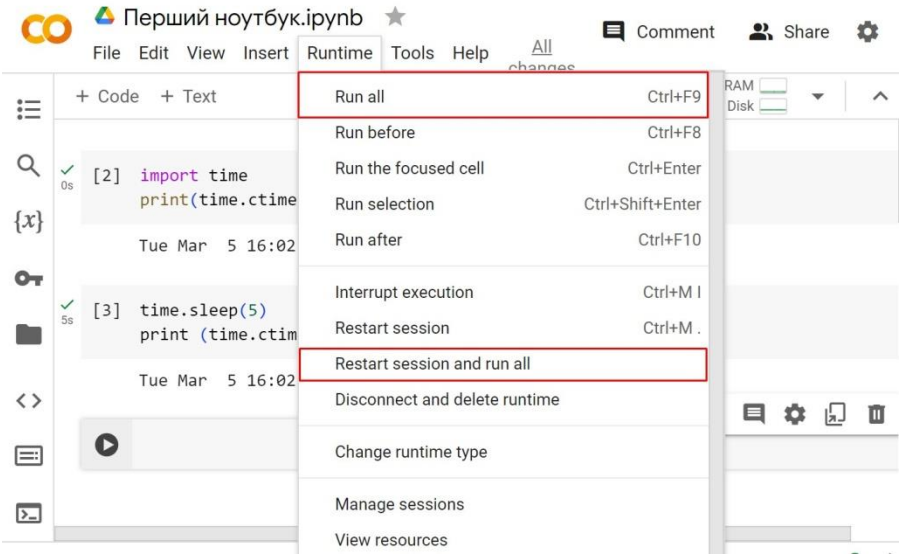
Нова комірка коду буде додана під поточною коміркою. Додайте наступні два рядки коду в новостворену комірку:

```
time.sleep(5)
print(time.ctime())
```

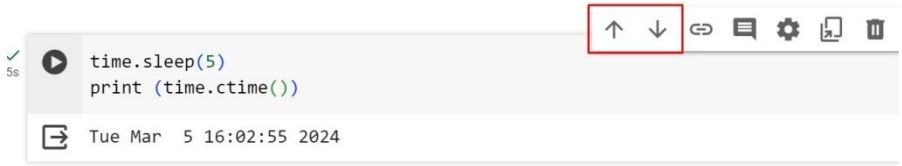
Тепер, якщо ви запустите цю комірку, ви побачите наступний результат:



Для запуску всіх комірок коду без переривань і покрокового натискання на кожну комірку вручну, виконайте пункт `Runtime \ Run all` або `Runtime \ Restart session and run all`:



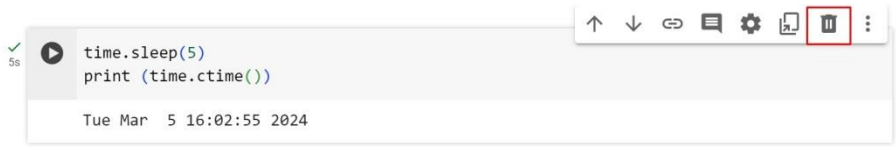
Якщо ваш блокнот містить велику кількість комірок із кодами, ви можете зіткнутися із ситуаціями, коли ви хочете змінити порядок виконання цих клітинок. Ви можете зробити це, виділивши комірку, яку ви хочете перемістити, і натиснувши на стрілки `Move cell up` або `Move cell down`, як показано на наступному скріншоті:



Ви можете натискати на стрілки кілька разів, щоб перемістити комірку більше, ніж на одну позицію.

Під час роботи з вашим проєктом ви могли б створити кілька непотрібних комірок у блокноті. Ви можете легко видалити такі комірки з проєкту одним

натиском миші. Натисніть на іконку із символом смітника у верхньому правому куті комірки з кодом для її видалення. Нижче показано приклад:



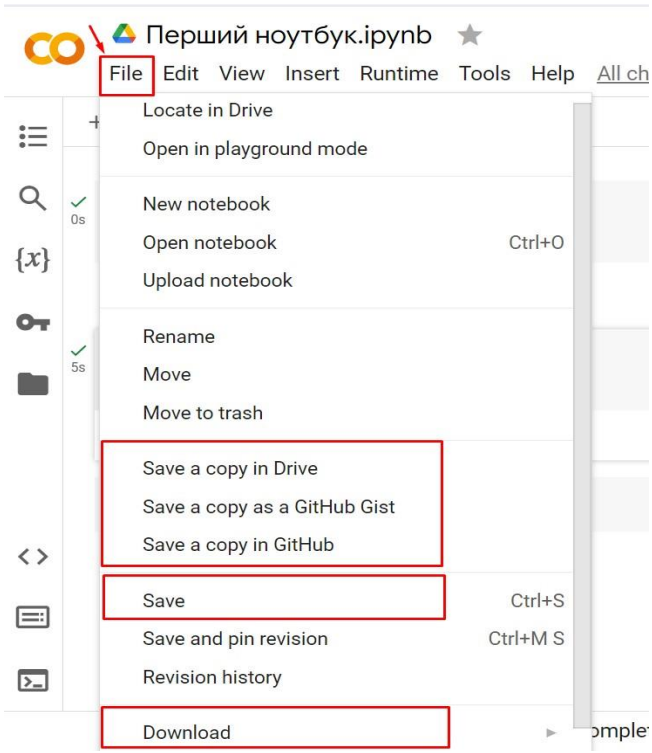
Для збереження вашого проєкту у вас є декілька опцій:

- зберегти його на сторінці GitHub;
- зберегти його у вашому Google диску;
- завантажити його на ваше локальне середовище у форматі `.ipynb`;
- зберегти його у хмарному середовищі Google Colab.

Для використання переваг усіх чотирьох опцій у меню даного середовище існує вкладка `File`, де можна натиснути на слідуючі пункти:

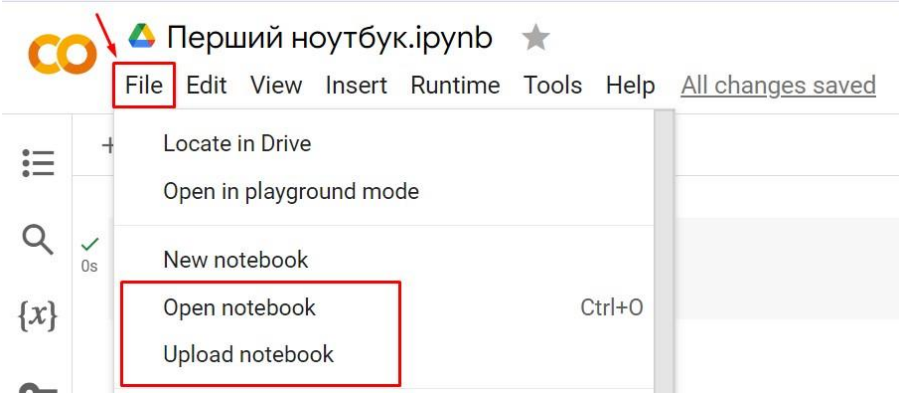
- `Save a copy in Drive` для збереження на Google диску;
- `Save a copy in GitHub` для збереження в репозитарії GitHub;
- `Save` для збереження копії у хмарному середовищі;
- `Download` для збереження у форматі `.ipynb` або класичному Python форматі `.py`.

Дані опції проілюстровано на скріншоті нижче:

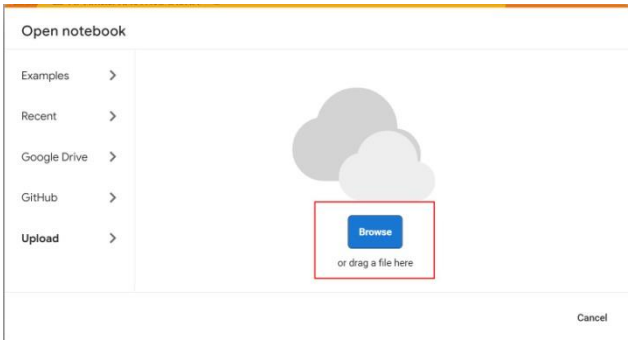


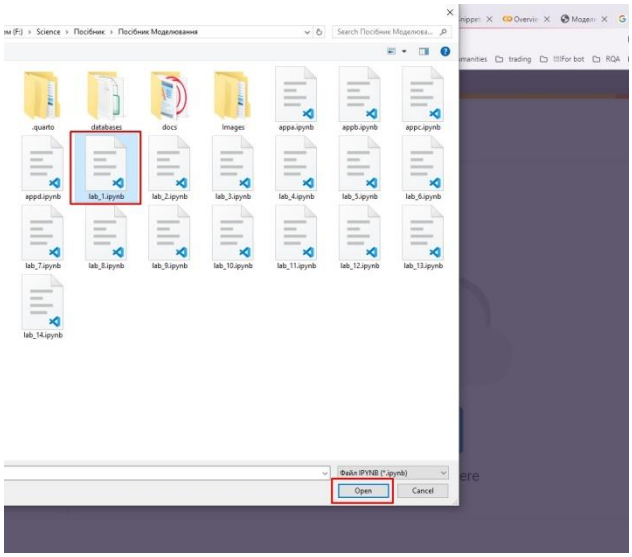
У подальшому може виникунути необхідність, наприклад, працювати не з власними файлами “з абсолютного нуля”, а, наприклад, уже готовими Jupyter блокнотами. Для цього треба перейти до `File \ Open notebook` або `File \ Upload`

`notebook`. Якщо ви обираєте пункт `File \ Open notebook`, тоді це означатиме, що ви бажаєте відкрити робочий блокнот із хмарного середовище або вашого Google диску. Якщо ви натискаєте `File \ Upload notebook`, ви хочете завантажити готовий блокнот із вашого персонального комп'ютера. Нижче представлено скріншот даних опцій:

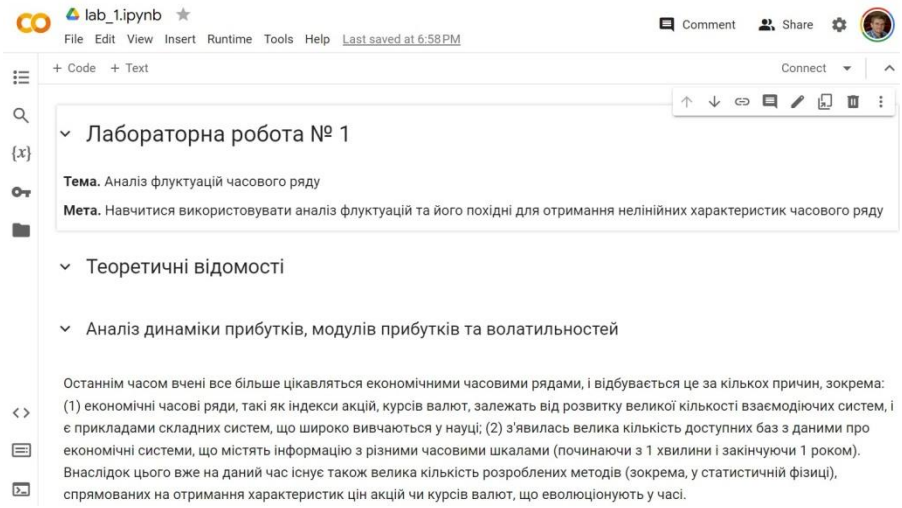


Завантажимо готовий блокнот однієї з лабораторних із моделювання в Python. Для цього натиснемо `File \ Upload notebook`. Далі нам треба буде натиснути на кнопку **Browse** та обрати потрібний нам файл, натиснувши на клавішу **Open**:



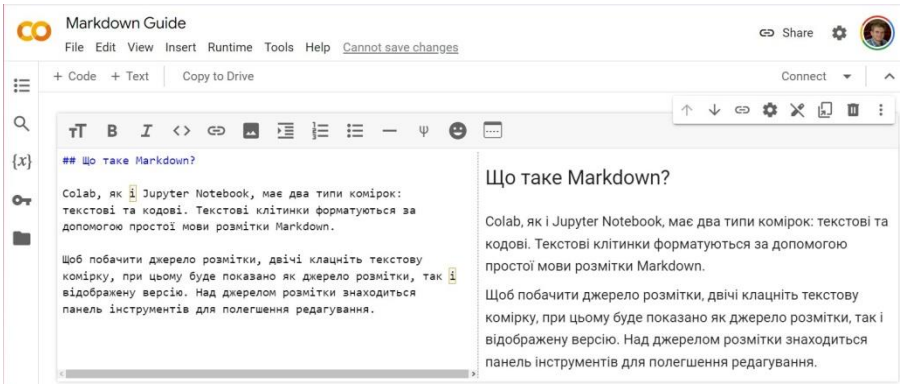


В якості прикладу ми обрали першу лабораторну роботу. Після завантаження цього файлу до Google Colab має з'явитись наступне вікно:



Colab, як і Jupyter Notebook, має два типи комірок: текстові та кодові. Текстові клітинки форматуються за допомогою простої мови розмітки Markdown.

Щоб побачити джерело розмітки, двічі клацніть текстову комірку. При цьому буде показано як джерело розмітки, так і відображену версію. Над джерелом розмітки знаходиться панель інструментів для полегшення редагування.



D.2 Синтаксис Markdown

Markdown	Вид
`**жирний текст**`	**жирний текст**
`*курсивний текст*` або `_курсивний текст_`	*курсивний текст*
`` `Виділення` ``	`Виділення`
`~~прочерк~~`	~~прочерк~~
`[Посилання](https://www.google.com)`	Посилання
`![Рисунок](https://www.google.com/images/rss.png)`	

Заголовки генеруються у вигляді назв секцій.

```
# Секція 1
# Секція 2
## Під-секція для Секції 2
### Під-секція під під-секцією під Секцією 2
# Секція 3
```

Зміст, доступний у лівій частині Colab, заповнюється з використанням не більше однієї назви розділу з кожної текстової комірки.

```
>Один рівень відступу
```

> Один рівень відступу

```
>>Два рівні відступу
```

> > Два рівні відступу

Блоки коду:

````
```{python}
print("a")
```

print("a")
````

Упорядковані списки:

```
1. Один
1. Два
1. Три
```

1. Один
2. Два
3. Три

Невпорядковані списки:

```
* Один
* Два
* Три
```

- Один
- Два
- Три

Рівняння:

```
$y=x^2$

$e^{i\pi} + 1 = 0$

$e^x=\sum_{i=0}^\infty \frac{1}{i!}x^i$

$\frac{n!}{k!(n-k)!} = {n \choose k}$
```

$$y = x^2$$

$$e^{i\pi} + 1 = 0$$

$$e^x = \sum_{i=0}^{\infty} \frac{1}{i!} x^i$$

$$\frac{n!}{k!\,(n-k)!} = \binom{n}{k}$$

Таблиці:

```
Назва першої колонки  | Назва другої колонки
----------------------|-----------------
Ряд 1, Колонка 1      | Ряд 1, Колонка 2
Ряд 2, Колонка 1      | Ряд 2, Колонка 2
```

Назва першої колонки	Назва другої колонки
Ряд 1, Колонка 1	Ряд 1, Колонка 2
Ряд 2, Колонка 1	Ряд 2, Колонка 2

Горизонтальне розділювання:

```
---
```

D.3 Різниця між Colab Markdown та іншими діалектами Mardown

Colab використовує marked.js і тому схожий, але не зовсім ідентичний Markdown, що використовується Jupyter і Github.

Colab підтримує (MathJax) LaTeX рівняння, як і Jupyter, але не дозволяє HTML-теги в Markdown. Colab не підтримує деякі доповнення GitHub, такі як смайлики і прапорці.

Якщо у блокноті Colab необхідно включити HTML, зверніться до %%html magic.

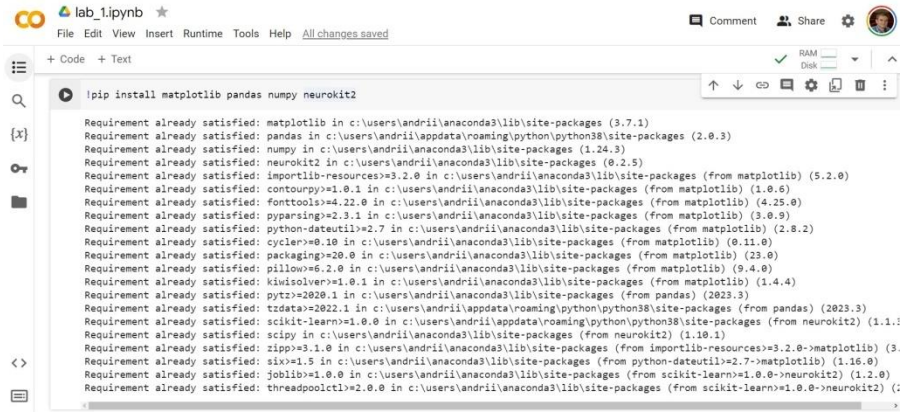
D.4 Встановлення відсутніх у Google Colab бібліотек

Щоб імпортувати бібліотеку, якої за замовчуванням немає у Collaboratory, ви можете скористатися командами `!pip install` або `!apt-get install`.

```
[1]  !pip install matplotlib-venn

Requirement already satisfied: matplotlib-venn in /usr/local/lib/python3.1
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.1
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-p
```

```
[2]  !apt-get -qq install -y libfluidsynth1

E: Package 'libfluidsynth1' has no installation candidate
```

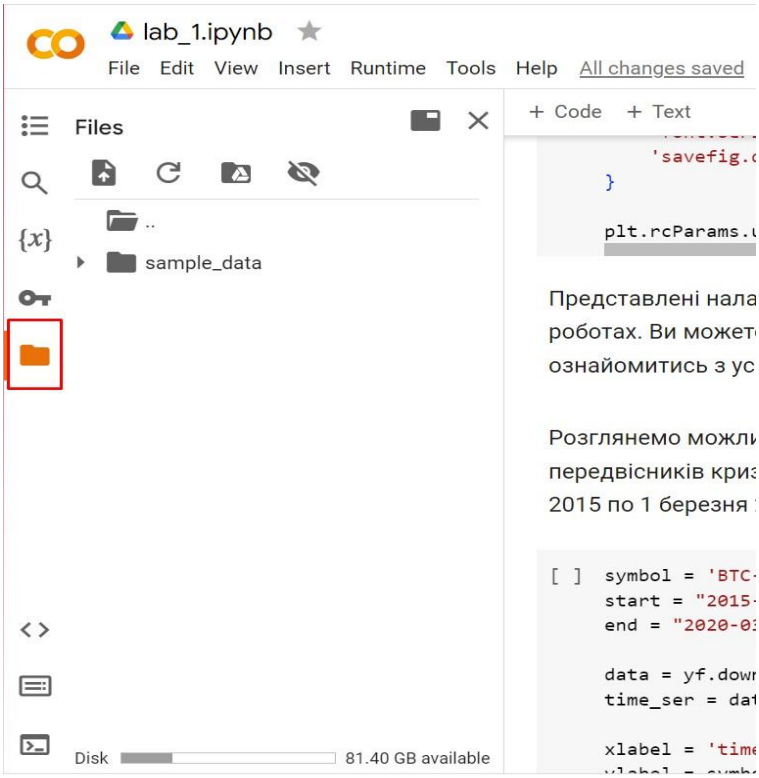
Нижче представлено приклад встановлення одразу чотирьох бібліотек: `matplotlib`, `pandas`, `numpy` і `neurokit2`:



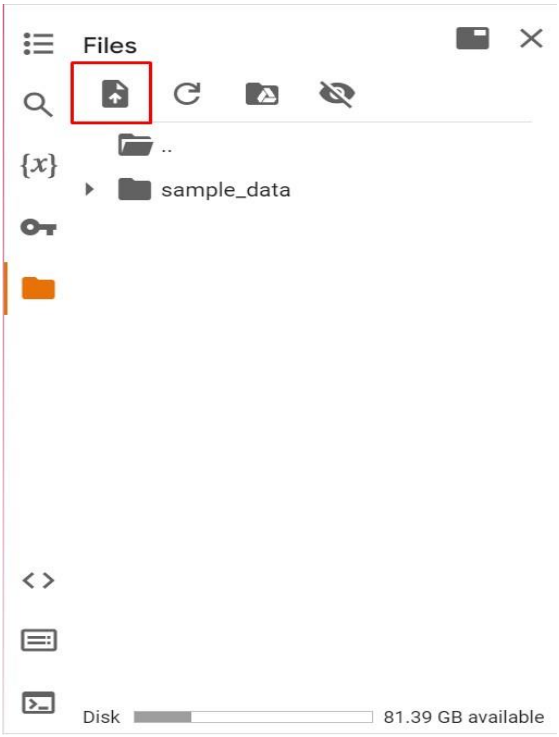
> ⓘ Примітка по встановленню бібліотек
>
> Кожного разу, коли ви відкриваєте Colab блокнот, ви розпочинаєте нову робочу сесію з чистого аркуша. Тобто, всі файли та бібліотеки, які ви завантажували під час попередньої сесії, будуть очищені. Тому при роботі з кожним новим файлом вам треба буде завантажувати всі необхідні бібліотеки із самого початку. У цьому й полягає головний недолік Google Colab у порівнянні з Jupyter Notebook

D.5 Імпорт власних файлів до Google Colab

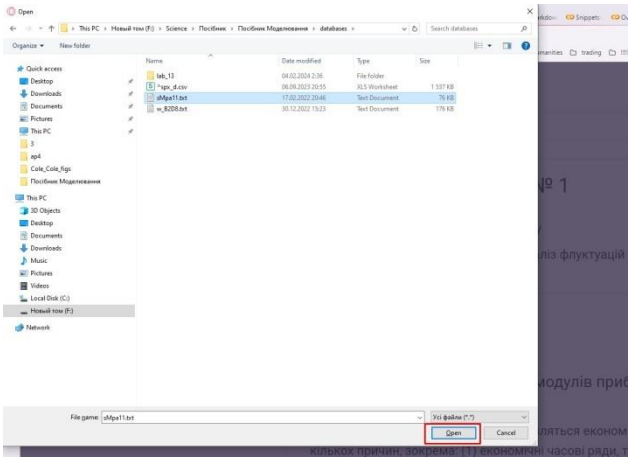
Буває так, що виникає необхідність у роботі з локальними файлами у форматі `.txt` або `.csv`, які можуть містити результати ваших власних вимірювань. Тоді потребується імпортувати їх до хмарного середовища Colab. Для цього переходимо до панелі меню з лівої сторони вашого середовища й натискаємо на пункт `Files`:

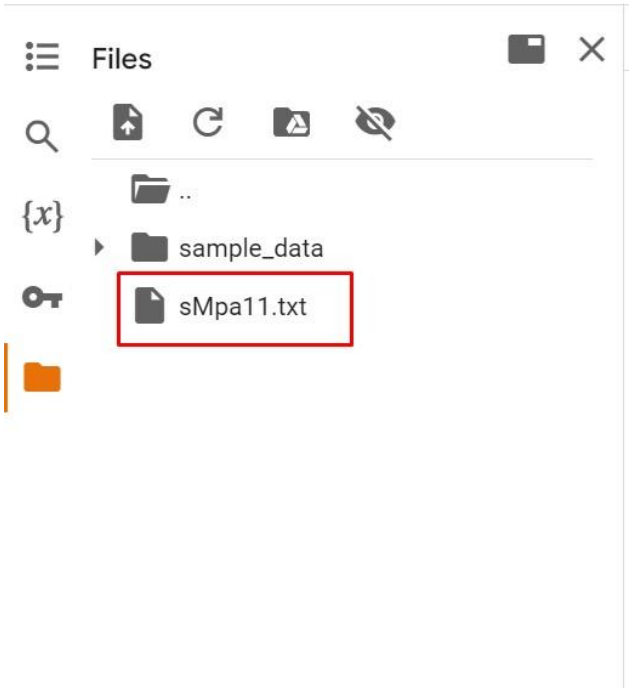


Бачимо, що за замовчуванням Google Colab відображає вміст вашого хмарного середовища та перелік системних файлів. Для імпорту власного файлу треба натиснути на перший значок із чотирьох, що зв’явилися в розгорнотому вікні. Потрібний для натиску значок виділений на скріншоті нижче:

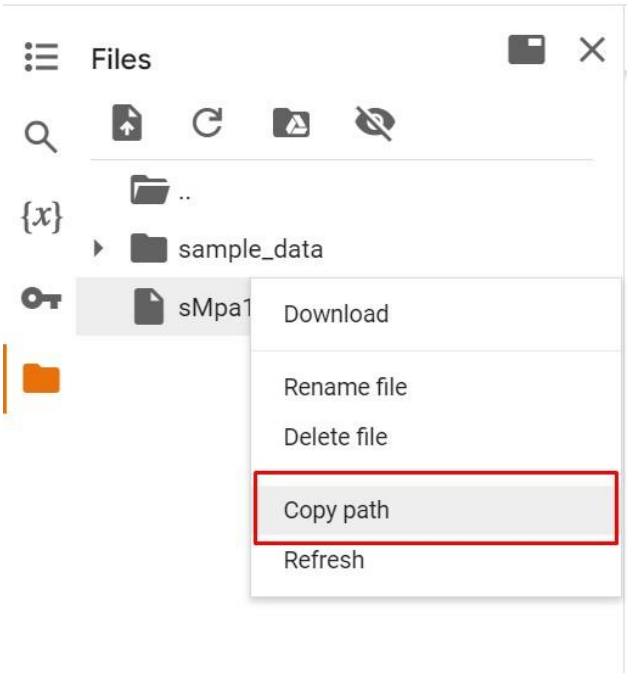


Після вибору потрібного для роботи файлу тиснемо на `Open`. У списку файлів має з'явитися обраний нами файл:





Якщо, наприклад, потребується зчитати даний файл із використанням методу `pd.read_csv()` бібліотеки Pandas, нам треба скопіювати шлях до зчитаного файлу. Для цього виділяємо файл курсором миші й натискаємо на вспливаючу трикрапку. Там має з'явитися пункт `Copy path`. Натиснувши на нього, можна підставляти шлях до методу змінної `path`, яка далі підставляється до методу `pd.read_csv()`:




```
[8]  symbol = 'sMpa11'                # Символ індексу

     path = "/content/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 Ox
     ylabel = symbol              # підпис по вісі Oy
```

Appendix E — Список рекомендованої літератури

[1] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw, *Geometry from a Time Series*, Phys. Rev. Lett. **45**, 712 (1980).

[2] F. Takens, *Detecting Strange Attractors in Turbulence*, in *Dynamical Systems and Turbulence, Warwick 1980*, edited by D. Rand and L.-S. Young (Springer Berlin Heidelberg, Berlin, Heidelberg, 1981), pp. 366–381.

[3] J.-P. Eckmann, S. O. Kamphorst, and D. Ruelle, *Recurrence Plots of Dynamical Systems*, Europhysics Letters **4**, 973 (1987).

[4] V. N. Soloviev and A. Belinskyi, *Methods of Nonlinear Dynamics and the Construction of Cryptocurrency Crisis Phenomena Precursors*, in *Proceedings of the 14th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops, Kyiv, Ukraine, May 14-17, 2018*, edited by V. Ermolayev, M. C. Suárez-Figueroa, V. Yakovyna, V. S. Kharchenko, V. Kobets, H. Kravtsov, V. S. Peschanenko, Y. Prytula, M. S. Nikitchenko, and A. Spivakovsky, Vol. 2104 (CEUR-WS.org, 2018), pp. 116–127.

[5] V. N. Soloviev and A. Belinskiy, *Complex Systems Theory and Crashes of Cryptocurrency Market*, in *Information and Communication Technologies in Education, Research, and Industrial Applications*, edited by V. Ermolayev, M. C. Suárez-Figueroa, V. Yakovyna, H. C. Mayr, M. Nikitchenko, and A. Spivakovsky (Springer International Publishing, Cham, 2019), pp. 276–297.

[6] K. Shockley and M. Riley, *In Recurrence Quantification Analysis: Theory and Best Practices*, 1st ed. (Springer, New York, 2015).

[7] T. Rawald, Scalable and Efficient Analysis of Large High-Dimensional Data Sets in the Context of Recurrence Analysis, PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät, 2018.

[8] A. M. Fraser and H. L. Swinney, *Independent Coordinates for Strange Attractors from Mutual Information*, Phys. Rev. A **33**, 1134 (1986).

[9] J. Theiler, *Statistical Precision of Dimension Estimators*, Phys. Rev. A **41**, 3038 (1990).

[10] M. Casdagli, S. Eubank, J. D. Farmer, and J. Gibson, *State Space Reconstruction in the Presence of Noise*, Physica D: Nonlinear Phenomena **51**, 52 (1991).

[11] M. T. Rosenstein, J. J. Collins, and C. J. De Luca, *A Practical Method for Calculating Largest Lyapunov Exponents from Small Data Sets*, Physica D: Nonlinear Phenomena **65**, 117 (1993).

[12] M. T. Rosenstein, J. J. Collins, and C. J. De Luca, *Reconstruction Expansion as a Geometry-Based Framework for Choosing Proper Delay Times*, Physica D: Nonlinear Phenomena **73**, 82 (1994).
[13] H. S. Kim, R. Eykholt, and J. D. Salas, *Nonlinear Dynamics, Delay Times, and Embedding Windows*, Physica D: Nonlinear Phenomena **127**, 48 (1999).
[14] J. V. Lyle, M. Nandi, and P. J. Aston, *Symmetric Projection Attractor Reconstruction: Sex Differences in the ECG*, Frontiers in Cardiovascular Medicine **8**, (2021).
[15] T. Gautama, D. Mandic, and M. Van Hulle, *A Differential Entropy Based Method for Determining the Optimal Embedding Parameters of a Signal*, Proceedings **6**, 29 (2003).
[16] P. Grassberger and I. Procaccia, *Measuring the Strangeness of Strange Attractors*, Physica D: Nonlinear Phenomena **9**, 189 (1983).
[17] P. Grassberger and I. Procaccia, *Characterization of Strange Attractors*, Phys. Rev. Lett. **50**, 346 (1983).
[18] P. Grassberger, *Generalized Dimensions of Strange Attractors*, Physics Letters A **97**, 227 (1983).
[19] M. B. Kennel, R. Brown, and H. D. I. Abarbanel, *Determining Embedding Dimension for Phase-Space Reconstruction Using a Geometrical Construction*, Phys. Rev. A **45**, 3403 (1992).
[20] A. Krakovská, K. Mezeiová, and H. Budáčová, *Use of False Nearest Neighbours for Selecting Variables and Embedding Parameters for State Space Reconstruction*, Journal of Complex Systems **2015**, (2015).
[21] C. Rhodes and M. Morari, *The False Nearest Neighbors Algorithm: An Overview*, Computers & Chemical Engineering **21**, S1149 (1997).
[22] S. G. Stavrinides et al., *On the Chaotic Nature of Random Telegraph Noise in Unipolar RRAM Memristor Devices*, Chaos, Solitons & Fractals **160**, 112224 (2022).
[23] L. Cao, *Practical Method for Determining the Minimum Embedding Dimension of a Scalar Time Series*, Physica D: Nonlinear Phenomena **110**, 43 (1997).
[24] C. L. Webber and J. P. Zbilut, *Dynamical Assessment of Physiological Systems and States Using Recurrence Plot Strategies*, Journal of Applied Physiology **76**, 965 (1994).
[25] J. P. Zbilut and C. L. Webber, *Embeddings and Delays as Derived from Quantification of Recurrence Plots*, Physics Letters A **171**, 199 (1992).
[26] N. Marwan, N. Wessel, U. Meyerfeldt, A. Schirdewan, and J. Kurths, *Recurrence-Plot-Based Measures of Complexity and Their Application to Heart-Rate-Variability Data*, Phys. Rev. E **66**, 026702 (2002).

[27] A. O. Bielinskyi, V. N. Soloviev, V. Solovieva, S. O. Semerikov, and M. A. Radin, *Recurrence Quantification Analysis of Energy Market Crises: A Nonlinear Approach to Risk Management*, in *Proceedings of the Selected and Revised Papers of 10th International Conference on Monitoring, Modeling & Management of Emergent Economy (M3E2-MLPEED 2022), Virtual Event, Kryvyi Rih, Ukraine, November 17-18, 2022*, edited by H. B. Danylchuk and S. O. Semerikov, Vol. 3465 (CEUR-WS.org, 2022), pp. 110–131.
[28] A. Tomashin, G. Leonardi, and S. Wallot, *Four Methods to Distinguish Between Fractal Dimensions in Time Series Through Recurrence Quantification Analysis*, Entropy **24**, (2022).
[29] M. S. Kanwal, J. A. Grochow, and N. Ay, *Comparing Information-Theoretic Measures of Complexity in Boltzmann Machines*, Entropy **19**, (2017).
[30] A. N. Kolmogorov, *Three Approaches to the Quantitative Definition of Information*, International Journal of Computer Mathematics **2**, 157 (1968).
[31] D. G. Bonchev, *Information Theoretic Complexity Measures*, in *Encyclopedia of Complexity and Systems Science*, edited by R. A. Meyers (Springer New York, New York, NY, 2009), pp. 4820–4839.
[32] L. T. Lui, G. Terrazas, H. Zenil, C. Alexander, and N. Krasnogor, *Complexity Measurement Based on Information Theory and Kolmogorov Complexity*, Artificial Life **21**, 205 (2015).
[33] M. Li and P. Vitányi, *Preliminaries*, in *An Introduction to Kolmogorov Complexity and Its Applications* (Springer New York, New York, NY, 2008), pp. 1–99.
[34] C. E. Shannon, *A Mathematical Theory of Communication*, Bell System Technical Journal **27**, 379 (1948).
[35] A. Lempel and J. Ziv, *On the Complexity of Finite Sequences*, IEEE Transactions on Information Theory **22**, 75 (1976).
[36] J.-L. Blanc, L. Pezard, and A. Lesne, *Delay Independence of Mutual-Information Rate of Two Symbolic Sequences*, Phys. Rev. E **84**, 036214 (2011).
[37] S. Zozor, P. Ravier, and O. Buttelli, *On Lempel–Ziv Complexity for Multidimensional Data Analysis*, Physica A: Statistical Mechanics and Its Applications **345**, 285 (2005).
[38] E. Estevez-Rams, R. Lora Serrano, B. Aragón Fernández, and I. Brito Reyes, *On the non-randomness of maximum Lempel Ziv complexity sequences of finite size*, Chaos: An Interdisciplinary Journal of Nonlinear Science **23**, 023118 (2013).

[39] R. Giglio, R. Matsushita, A. Figueiredo, I. Gleria, and S. D. Silva, *Algorithmic Complexity Theory and the Relative Efficiency of Financial Markets*, Europhysics Letters **84**, 48005 (2008).
[40] C. Taufemback, R. Giglio, and S. D. Silva, *Algorithmic complexity theory detects decreases in the relative efficiency of stock markets in the aftermath of the 2008 financial crisis*, Economics Bulletin **31**, 1631 (2011).
[41] R. Giglio and S. Da Silva, Ranking the Stocks Listed on Bovespa According to Their Relative Efficiency, MPRA Paper, University Library of Munich, Germany, 2009.
[42] Y. Bai, Z. Liang, and X. Li, *A Permutation Lempel-Ziv Complexity Measure for EEG Analysis*, Biomedical Signal Processing and Control **19**, 102 (2015).
[43] M. Borowska, *Multiscale Permutation Lempel–Ziv Complexity Measure for Biomedical Signal Analysis: Interpretation and Application to Focal EEG Signals*, Entropy **23**, (2021).
[44] B. K. Hillen, G. T. Yamaguchi, J. J. Abbas, and R. Jung, *Joint-Specific Changes in Locomotor Complexity in the Absence of Muscle Atrophy Following Incomplete Spinal Cord Injury*, Journal of NeuroEngineering and Rehabilitation **10**, 1 (2013).
[45] M. D. Costa, C.-K. Peng, and A. L. Goldberger, *Multiscale Analysis of Heart Rate Dynamics: Entropy and Time Irreversibility Measures*, Cardiovascular Engineering **8**, 88 (2008).
[46] R. A. Fisher and E. J. Russell, *On the Mathematical Foundations of Theoretical Statistics*, Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character **222**, 309 (1922).
[47] B. Hjorth, *EEG Analysis Based on Time Domain Properties*, Electroencephalography and Clinical Neurophysiology **29**, 306 (1970).
[48] F. Mormann, T. Kreuz, C. Rieke, R. G. Andrzejak, A. Kraskov, P. David, C. E. Elger, and K. Lehnertz, *On the Predictability of Epileptic Seizures*, Clinical Neurophysiology **116**, 569 (2005).
[49] V. Marmelat, K. Torre, and D. Delignieres, *Relative Roughness: An Index for Testing the Suitability of the Monofractal Model*, Frontiers in Physiology **3**, (2012).
[50] T. M. Cover, *Elements of Information Theory* (John Wiley & Sons, 1999).
[51] A. Hacine-Gharbi and P. Ravier, *A Binning Formula of Bi-Histogram for Joint Entropy Estimation Using Mean Square Error Minimization*, Pattern Recognition Letters **101**, 21 (2018).
[52] R. Clausius, T. A. Hirst, and J. Tyndall, *The Mechanical Theory of Heat: With Its Applications to the Steam-Engine and to the Physical Properties of Bodies* (J. Van Voorst, 1867).

[53] L. Boltzmann, *Weitere Studien über Das wärmegleichgewicht Unter Gasmolekülen*, in *Kinetische Theorie II: Irreversible Prozesse Einführung Und Originaltexte* (Vieweg+Teubner Verlag, Wiesbaden, 1970), pp. 115–225.
[54] S. M. Pincus, I. M. Gladstone, and R. A. Ehrenkranz, *A Regularity Statistic for Medical Data Analysis*, Journal of Clinical Monitoring **7**, 335 (1991).
[55] S. M. Pincus, *Approximate Entropy as a Measure of System Complexity*, Proceedings of the National Academy of Sciences **88**, 2297 (1991).
[56] V. N. Soloviev, A. O. Bielinskyi, and N. A. Kharadzjan, *Coverage of the Coronavirus Pandemic Through Entropy Measures*, in *3rd Workshop for Young Scientists in Computer Science and Software Engineering (CS and SE and SW 2020), Kryvyi Rih, Ukraine, November 27, 2020*, edited by A. E. Kiv, S. O. Semerikov, V. N. Soloviev, and A. M. Striuk, Vol. 2832 (CEUR-WS.org, 2021), pp. 24–42.
[57] W. Chen, Z. Wang, H. Xie, and W. Yu, *Characterization of Surface EMG Signal Based on Fuzzy Entropy*, IEEE Transactions on Neural Systems and Rehabilitation Engineering **15**, 266 (2007).
[58] H.-B. Xie, W.-X. He, and H. Liu, *Measuring Time Series Regularity Using Nonlinear Similarity-Based Sample Entropy*, Physics Letters A **372**, 7140 (2008).
[59] A. O. Bielinskyi, V. N. Soloviev, S. O. Semerikov, and V. V. Solovieva, *IDENTIFYING STOCK MARKET CRASHES BY FUZZY MEASURES OF COMPLEXITY*, Neuro-Fuzzy Modeling Techniques in Economics **10**, 3 (2021).
[60] J. S. Richman and J. R. Moorman, *Physiological Time-Series Analysis Using Approximate Entropy and Sample Entropy*, American Journal of Physiology-Heart and Circulatory Physiology **278**, H2039 (2000).
[61] C. Bandt and B. Pompe, *Permutation Entropy: A Natural Complexity Measure for Time Series*, Phys. Rev. Lett. **88**, 174102 (2002).
[62] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis* (Cambridge University Press, 2004).
[63] V. N. Soloviev, A. Bielinskyi, and V. Solovieva, *Entropy Analysis of Crisis Phenomena for DJIA Index*, in *Proceedings of the 15th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops, Kherson, Ukraine, June 12-15, 2019*, edited by V. Ermolayev, F. Mallet, V. Yakovyna, V. S. Kharchenko, V. Kobets, A. Kornilowicz, H. Kravtsov, M. S. Nikitchenko, S. Semerikov, and A. Spivakovsky, Vol. 2393 (CEUR-WS.org, 2019), pp. 434–449.
[64] S. J. Roberts, W. Penny, and I. Rezek, *Temporal and Spatial Complexity Measures for Electroencephalogram Based Brain-Computer Interfacing*, Medical & Biological Engineering & Computing **37**, 93 (1999).

[65] M. Rostaghi and H. Azami, *Dispersion Entropy: A Measure for Time-Series Analysis*, IEEE Signal Processing Letters **23**, 610 (2016).
[66] J. C. Crepeau and L. K. Isaacson, Journal of Non-Equilibrium Thermodynamics **16**, 137 (1991).
[67] B. B. Mandelbrot and B. B. Mandelbrot, *The Fractal Geometry of Nature*, Vol. 1 (WH freeman New York, 1982).
[68] B. B. Mandelbrot, C. J. G. Evertsz, and Y. Hayakawa, *Exactly Self-Similar Left-Sided Multifractal Measures*, Phys. Rev. A **42**, 4528 (1990).
[69] H. F. Jelinek, N. Elston, and B. Zietsch, *Fractal Analysis: Pitfalls and Revelations in Neuroscience*, in *Fractals in Biology and Medicine*, edited by G. A. Losa, D. Merlini, T. F. Nonnenmacher, and E. R. Weibel (Birkhäuser Basel, Basel, 2005), pp. 85–94.
[70] H. Steinhaus, *Length, Shape and Area*, in *Colloquium Mathematicum*, Vol. 3 (Polska Akademia Nauk. Instytut Matematyczny PAN, 1954), pp. 1–13.
[71] A. Vulpiani, *Lewis Fry Richardson: Scientist, Visionary and Pacifist*, Lettera Matematica **2**, 121 (2014).
[72] B. Hayes, *Computing Science: Statistics of Deadly Quarrels*, American Scientist **90**, 10 (2002).
[73] B. Mandelbrot, *How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension*, Science **156**, 636 (1967).
[74] S. V. Bozhokin and D. A. Parshin, *Fractals and Multifractals: Textbook* (Scientific; Publishing Center "Regular; Chaotic Dynamics", 2001).
[75] T. Gneiting, H. Ševčíková, and D. B. Percival, *Estimators of Fractal Dimension: Assessing the Roughness of Time Series and Spatial Data*, Statistical Science **27**, 247 (2012).
[76] K. Falconer, *Fractal Geometry: Mathematical Foundations and Applications* (John Wiley & Sons, 2003).
[77] B. B. Mandelbrot and J. A. Wheeler, *The Fractal Geometry of Nature*, American Journal of Physics **51**, 286 (1983).
[78] H. E. Hurst, *Long-Term Storage Capacity of Reservoirs*, Transactions of the American Society of Civil Engineers **116**, 770 (1951).
[79] H. E. Hurst, *A Suggested Statistical Model of Some Time Series Which Occur in Nature*, Nature **180**, 494 (1957).
[80] C.-K. Peng, S. V. Buldyrev, S. Havlin, M. Simons, H. E. Stanley, and A. L. Goldberger, *Mosaic Organization of DNA Nucleotides*, Phys. Rev. E **49**, 1685 (1994).

[81] Z.-Q. Jiang, W.-J. Xie, and W.-X. Zhou, *Testing the Weak-Form Efficiency of the WTI Crude Oil Futures Market*, Physica A: Statistical Mechanics and Its Applications **405**, 235 (2014).
[82] T. Higuchi, *Approach to an Irregular Time Series on the Basis of the Fractal Theory*, Physica D: Nonlinear Phenomena **31**, 277 (1988).
[83] C. F. Vega and J. Noel, *Parameters Analyzed of Higuchi's Fractal Dimension for EEG Brain Signals*, in *2015 Signal Processing Symposium (SPSympo)* (2015), pp. 1–5.
[84] A. Petrosian, *Kolmogorov Complexity of Finite Sequences and Recognition of Different Preictal EEG Patterns*, in *Proceedings Eighth IEEE Symposium on Computer-Based Medical Systems* (1995), pp. 212–217.
[85] R. Esteller, G. Vachtsevanos, J. Echauz, and B. Litt, *A Comparison of Waveform Fractal Dimension Algorithms*, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications **48**, 177 (2001).
[86] C. Goh, B. Hamadicharef, G. T. Henderson, and E. C. Ifeachor, *Comparison of Fractal Dimension Algorithms for the Computation of EEG Biomarkers for Dementia*, in *2nd International Conference on Computational Intelligence in Medicine and Healthcare (CIMED2005)* (Professor José Manuel Fonseca, UNINOVA, Portugal, Lisbon, Portugal, 2005).
[87] M. J. Katz, *Fractals and the Analysis of Waveforms*, Computers in Biology and Medicine **18**, 145 (1988).
[88] C. Sevcik, *A Procedure to Estimate the Fractal Dimension of Waveforms*, (2010).
[89] A. Kalauzi, T. Bojić, and L. Rakić, *Extracting Complexity Waveforms from One-Dimensional Signals*, Nonlinear Biomedical Physics **3**, 1 (2009).
[90] A. Kalauzi, T. Bojić, and L. Rakić, *Extracting Complexity Waveforms from One-Dimensional Signals*, Nonlinear Biomedical Physics **3**, (2009).
[91] F. Hasselman, *When the Blind Curve Is Finite: Dimension Estimation and Model Inference Based on Empirical Waveforms*, Frontiers in Physiology **4**, (2013).
[92] R. F. Voss, *Fractals in Nature: From Characterization to Simulation*, in *The Science of Fractal Images*, edited by H.-O. Peitgen and D. Saupe (Springer New York, New York, NY, 1988), pp. 21–70.
[93] A. A. Anis and E. H. Lloyd, *The Expected Value of the Adjusted Rescaled Hurst Range of Independent Normal Summands*, Biometrika **63**, 111 (1976).
[94] T. C. Halsey, M. H. Jensen, L. P. Kadanoff, I. Procaccia, and B. I. Shraiman, *Fractal Measures and Their Singularities: The Characterization of Strange Sets*, Phys. Rev. A **33**, 1141 (1986).

[95] U. Frisch and G. Parisi, *Turbulence and Predictability of Geophysical Flows and Climate Dynamics*, in *Proceedings of the International School of Physics "enrico Fermi," Course LXXXVIII, Varenna, 1983* (North-Holland, New York, 1985).
[96] T. C. Halsey, M. H. Jensen, L. P. Kadanoff, I. Procaccia, and B. I. Shraiman, *Fractal Measures and Their Singularities: The Characterization of Strange Sets*, Nuclear Physics B - Proceedings Supplements **2**, 501 (1987).
[97] J. W. Kantelhardt, E. Koscielny-Bunde, H. H. A. Rego, S. Havlin, and A. Bunde, *Detecting Long-Range Correlations with Detrended Fluctuation Analysis*, Physica A: Statistical Mechanics and Its Applications **295**, 441 (2001).
[98] J. W. Kantelhardt, *Fractal and Multifractal Time Series*, in *Mathematics of Complexity and Dynamical Systems*, edited by R. A. Meyers (Springer New York, New York, NY, 2011), pp. 463–487.
[99] J. W. Kantelhardt, S. A. Zschiegner, E. Koscielny-Bunde, S. Havlin, A. Bunde, and H. E. Stanley, *Multifractal Detrended Fluctuation Analysis of Nonstationary Time Series*, Physica A: Statistical Mechanics and Its Applications **316**, 87 (2002).
[100] S. Dutta, *Multifractal Properties of ECG Patterns of Patients Suffering from Congestive Heart Failure*, Journal of Statistical Mechanics: Theory and Experiment **2010**, P12021 (2010).
[101] E. Maiorino, L. Livi, A. Giuliani, A. Sadeghian, and A. Rizzi, *Multifractal Characterization of Protein Contact Networks*, Physica A: Statistical Mechanics and Its Applications **428**, 302 (2015).
[102] P. H. Figueirêdo, E. Nogueira, M. A. Moret, and S. Coutinho, *Multifractal Analysis of Polyalanines Time Series*, Physica A: Statistical Mechanics and Its Applications **389**, 2090 (2010).
[103] G. R. Jafari, P. Pedram, and L. Hedayatifar, *Erratum: Long-Range Correlation and Multifractality in Bach's Inventions Pitches*, Journal of Statistical Mechanics: Theory and Experiment **2012**, E03001 (2012).
[104] Z.-Q. Jiang, W.-J. Xie, W.-X. Zhou, and D. Sornette, *Multifractal Analysis of Financial Markets: A Review*, Reports on Progress in Physics **82**, 125901 (2019).
[105] L. Telesca, V. Lapenna, and M. Macchiato, *Multifractal Fluctuations in Earthquake-Related Geoelectrical Signals*, New Journal of Physics **7**, 214 (2005).
[106] E. G. Yee Leung and Z. Yu, *Temporal Scaling Behavior of Avian Influenza a (H5N1): The Multifractal Detrended Fluctuation Analysis*, Annals of the Association of American Geographers **101**, 1221 (2011).
[107] F. Liao and Y.-K. Jan, *Using Multifractal Detrended Fluctuation Analysis to Assess Sacral Skin Blood Flow Oscillations in People with Spinal Cord Injury*, The Journal of Rehabilitation Research and Development **48**, 787 (2011).

[108] L. Telesca, V. Lapenna, and M. Macchiato, *Multifractal Fluctuations in Seismic Interspike Series*, Physica A: Statistical Mechanics and Its Applications **354**, 629 (2005).
[109] M. S. Movahed, F. Ghasemi, S. Rahvar, and M. R. R. Tabar, *Long-Range Correlation in Cosmic Microwave Background Radiation*, Phys. Rev. E **84**, 021103 (2011).
[110] P. Mali, S. Sarkar, S. Ghosh, A. Mukhopadhyay, and G. Singh, *Multifractal Detrended Fluctuation Analysis of Particle Density Fluctuations in High-Energy Nuclear Collisions*, Physica A: Statistical Mechanics and Its Applications **424**, 25 (2015).
[111] I. T. Pedron, *Correlation and Multifractality in Climatological Time Series*, Journal of Physics: Conference Series **246**, 012034 (2010).
[112] R. Rak, S. Drożdż, J. Kwapień, and P. Oświęcimka, *Detrended Cross-Correlations Between Returns, Volatility, Trading Activity, and Volume Traded for the Stock Market Companies*, Europhysics Letters **112**, 48001 (2015).
[113] M. Wątorek, S. Drożdż, J. Kwapień, L. Minati, P. Oświęcimka, and M. Stanuszek, *Multiscale Characteristics of the Emerging Global Cryptocurrency Market*, Physics Reports **901**, 1 (2021).
[114] C.-K. Peng, S. Havlin, H. E. Stanley, and A. L. Goldberger, *Quantification of Scaling Exponents and Crossover Phenomena in Nonstationary Heartbeat Time Series*, Chaos: An Interdisciplinary Journal of Nonlinear Science **5**, 82 (1995).
[115] E. Canessa, *Multifractality in Time Series*, Journal of Physics A: Mathematical and General **33**, 3637 (2000).
[116] A. Kasprzak, R. Kutner, J. Perelló, and J. Masoliver, *Higher-Order Phase Transitions on Financial Markets*, The European Physical Journal B: Condensed Matter and Complex Systems **76**, 513 (2010).
[117] M. Dai, C. Zhang, and D. Zhang, *Multifractal and Singularity Analysis of Highway Volume Data*, Physica A: Statistical Mechanics and Its Applications **407**, 332 (2014).
[118] M. Dai, J. Hou, and D. Ye, *Multifractal Detrended Fluctuation Analysis Based on Fractal Fitting: The Long-Range Correlation Detection Method for Highway Volume Data*, Physica A: Statistical Mechanics and Its Applications **444**, 722 (2016).
[119] X. Sun, H. Chen, Z. Wu, and Y. Yuan, *Multifractal Analysis of Hang Seng Index in Hong Kong Stock Market*, Physica A: Statistical Mechanics and Its Applications **291**, 553 (2001).
[120] E. A. Ihlen, *Introduction to Multifractal Detrended Fluctuation Analysis in Matlab*, Frontiers in Physiology **3**, (2012).

[121] P. Oświęcimka, L. Livi, and S. Drożdż, *Right-Side-Stretched Multifractal Spectra Indicate Small-Worldness in Networks*, Communications in Nonlinear Science and Numerical Simulation **57**, 231 (2018).
[122] S. Drożdż and P. Oświęcimka, *Detecting and Interpreting Distortions in Hierarchical Organization of Complex Time Series*, Phys. Rev. E **91**, 030902 (2015).
[123] S. Drożdż, R. Kowalski, P. Oświęcimka, R. Rak, and R. Gębarowski, *Dynamical Variety of Shapes in Financial Multifractality*, Complexity **2018**, 1 (2018).
[124] A. Orozco-Duque, D. Novak, V. Kremen, and J. Bustamante, *Multifractal Analysis for Grading Complex Fractionated Electrograms in Atrial Fibrillation*, Physiological Measurement **36**, 2269 (2015).
[125] A. Faini, G. Parati, and P. Castiglioni, *Multiscale Assessment of the Degree of Multifractality for Physiological Time Series*, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **379**, 20200254 (2021).
[126] A. Bielinskyi, V. Soloviev, V. Solovieva, A. Matviychuk, and S. Semerikov, *The Analysis of Multifractal Cross-Correlation Connectedness Between Bitcoin and the Stock Market*, in *Information Technology for Education, Science, and Technics*, edited by E. Faure, O. Danchenko, M. Bondarenko, Y. Tryus, C. Bazilo, and G. Zaspa (Springer Nature Switzerland, Cham, 2023), pp. 323–345.
[127] E. P. Wigner, *On the Statistical Distribution of the Widths and Spacings of Nuclear Resonance Levels*, Mathematical Proceedings of the Cambridge Philosophical Society **47**, 790 (1951).
[128] E. P. Wigner, *On a Class of Analytic Functions from the Quantum Theory of Collisions*, Annals of Mathematics **53**, 36 (1951).
[129] F. J. Dyson, *Statistical Theory of the Energy Levels of Complex Systems. I*, Journal of Mathematical Physics **3**, 140 (2004).
[130] F. J. Dyson and M. L. Mehta, *Statistical Theory of the Energy Levels of Complex Systems. IV*, Journal of Mathematical Physics **4**, 701 (2004).
[131] M. L. Mehta and F. J. Dyson, *Statistical Theory of the Energy Levels of Complex Systems. V*, Journal of Mathematical Physics **4**, 713 (2004).
[132] M. L. Mehta, *Random Matrices* (Academic Press, 1991).
[133] T. A. Brody, J. Flores, J. B. French, P. A. Mello, A. Pandey, and S. S. M. Wong, *Random-Matrix Physics: Spectrum and Strength Fluctuations*, Rev. Mod. Phys. **53**, 385 (1981).
[134] L. Laloux, P. Cizeau, J.-P. Bouchaud, and M. Potters, *Noise Dressing of Financial Correlation Matrices*, Phys. Rev. Lett. **83**, 1467 (1999).
[135] *Aspects of Multivariate Statistical Theory* (Wiley, New York, 1982).

[136] F. J. Dyson, *Distribution of Eigenvalues for a Class of Real Symmetric Matrices*, Revista Mexicana de Fisica **20**, 231 (1971).
[137] A. M. Sengupta and P. P. Mitra, *Distributions of Singular Values for Some Random Matrices*, Phys. Rev. E **60**, 3389 (1999).
[138] V. Plerou, P. Gopikrishnan, B. Rosenow, L. A. Nunes Amaral, and H. E. Stanley, *Universal and Nonuniversal Properties of Cross Correlations in Financial Time Series*, Phys. Rev. Lett. **83**, 1471 (1999).
[139] T. Guhr, A. Müller–Groeling, and H. A. Weidenmüller, *Random-Matrix Theories in Quantum Physics: Common Concepts*, Physics Reports **299**, 189 (1998).
[140] Y. V. Fyodorov and A. D. Mirlin, *Analytical Derivation of the Scaling Law for the Inverse Participation Ratio in Quasi-One-Dimensional Disordered Systems*, Phys. Rev. Lett. **69**, 1093 (1992).
[141] C. J. Gavilán-Moreno and G. Espinosa-Paredes, *Using Largest Lyapunov Exponent to Confirm the Intrinsic Stability of Boiling Water Reactors*, Nuclear Engineering and Technology **48**, 434 (2016).
[142] A. Prieto-Guerrero and G. Espinosa-Paredes, *Dynamics of BWRs and Mathematical Models*, in *Linear and Non-Linear Stability Analysis in Boiling Water Reactors*, edited by A. Prieto-Guerrero and G. Espinosa-Paredes (Woodhead Publishing, 2019), pp. 193–268.
[143] V. N. Soloviev, A. Bielinskyi, O. Serdyuk, V. Solovieva, and S. Semerikov, *Lyapunov Exponents as Indicators of the Stock Market Crashes*, in *Proceedings of the 16th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer. Volume II: Workshops, Kharkiv, Ukraine, October 06-10, 2020*, edited by O. Sokolov, G. Zholtkevych, V. Yakovyna, Y. Tarasich, V. Kharchenko, V. Kobets, O. Burov, S. Semerikov, and H. Kravtsov, Vol. 2732 (CEUR-WS.org, 2020), pp. 455–470.
[144] D. Nychka, S. Ellner, A. R. Gallant, and D. McCaffrey, *Finding Chaos in Noisy Systems*, Journal of the Royal Statistical Society. Series B (Methodological) **54**, 399 (1992).
[145] A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano, *Determining Lyapunov Exponents from a Time Series*, Physica D: Nonlinear Phenomena **16**, 285 (1985).
[146] M. Sano and Y. Sawada, *Measurement of the Lyapunov Spectrum from a Chaotic Time Series*, Phys. Rev. Lett. **55**, 1082 (1985).
[147] J.-P. Eckmann, S. O. Kamphorst, D. Ruelle, and S. Ciliberto, *Liapunov Exponents from Time Series*, Phys. Rev. A **34**, 4971 (1986).
[148] U. Parlitz, *Identification of True and Spurious Lyapunov Exponents from Time Series*, International Journal of Bifurcation and Chaos **02**, 155 (1992).

[149] M. Balcerzak, D. Pikunov, and A. Dabrowski, *The Fastest, Simplified Method of Lyapunov Exponents Spectrum Estimation for Continuous-Time Dynamical Systems*, Nonlinear Dynamics **94**, 3053 (2018).
[150] H. D. I. Abarbanel, R. Brown, J. J. Sidorowich, and L. Sh. Tsimring, *The Analysis of Observed Chaotic Data in Physical Systems*, Rev. Mod. Phys. **65**, 1331 (1993).
[151] J.-P. Eckmann and D. Ruelle, *Ergodic Theory of Chaos and Strange Attractors*, Rev. Mod. Phys. **57**, 617 (1985).
[152] A. Bielinskyi, S. Semerikov, O. Serdyuk, V. Solovieva, V. N. Soloviev, and L. Pichl, *Econophysics of Sustainability Indices*, in *Proceedings of the Selected Papers of the Special Edition of International Conference on Monitoring, Modeling & Management of Emergent Economy (M3E2-MLPEED 2020), Odessa, Ukraine, July 13-18, 2020*, edited by A. Kiv, Vol. 2713 (CEUR-WS.org, 2020), pp. 372–392.
[153] G. Nicolis, I. Prigogine, W. H. Freeman, and Company, *Exploring Complexity: An Introduction* (W.H. Freeman, 1989).
[154] C. Tsallis, *Possible Generalization of Boltzmann-Gibbs Statistics*, Journal of Statistical Physics **52**, 479 (1988).
[155] C. Tsallis, *Dynamical Scenario for Nonextensive Statistical Mechanics*, Physica A: Statistical Mechanics and Its Applications **340**, 1 (2004).
[156] C. Tsallis, M. Gell-Mann, and Y. Sato, *Asymptotically Scale-Invariant Occupancy of Phase Space Makes the Entropy S_q Extensive*, Proceedings of the National Academy of Sciences **102**, 15377 (2005).
[157] C. Tsallis, *Economics and Finance: Q-Statistical Stylized Features Galore*, Entropy **19**, (2017).
[158] C. Tsallis, *Beyond Boltzmann–Gibbs–Shannon in Physics and Elsewhere*, Entropy **21**, (2019).
[159] E. G. Pavlos, O. E. Malandraki, O. V. Khabarova, L. P. Karakatsanis, G. P. Pavlos, and G. Livadiotis, *Non-Extensive Statistical Analysis of Energetic Particle Flux Enhancements Caused by the Interplanetary Coronal Mass Ejection-Heliospheric Current Sheet Interaction*, Entropy **21**, (2019).
[160] R. de Oliveira, S. Brito, L. da Silva, and C. Tsallis, *Connecting Complex Networks to Nonadditive Entropies*, Scientific Reports **11**, 1130 (2021).
[161] G. Pavlos, A. Iliopoulos, L. Karakatsanis, M. Xenakis, and E. Pavlos, *Complexity of Economical Systems.*, Journal of Engineering Science & Technology Review **8**, (2015).
[162] G. L. Ferri, M. F. Reynoso Savio, and A. Plastino, *Tsallis' q-Triplet and the Ozone Layer*, Physica A: Statistical Mechanics and Its Applications **389**, 1829 (2010).

[163] S. Umarov, C. Tsallis, and S. Steinberg, *On Aq-Central Limit Theorem Consistent with Nonextensive Statistical Mechanics*, Milan Journal of Mathematics **76**, 307 (2008).
[164] C. Anteneodo and C. Tsallis, *Breakdown of Exponential Sensitivity to Initial Conditions: Role of the Range of Interactions*, Phys. Rev. Lett. **80**, 5313 (1998).
[165] C. TSALLIS, *Some Open Problems in Nonextensive Statistical Mechanics*, International Journal of Bifurcation and Chaos **22**, 1230030 (2012).
[166] D. Stosic, D. Stosic, T. B. Ludermir, and T. Stosic, *Nonextensive Triplets in Cryptocurrency Exchanges*, Physica A: Statistical Mechanics and Its Applications **505**, 1069 (2018).
[167] A. O. Bielinskyi, A. V. Matviychuk, O. A. Serdyuk, S. O. Semerikov, V. V. Solovieva, and V. N. Soloviev, *Correlational and Non-Extensive Nature of Carbon Dioxide Pricing Market*, in *ICTERI 2021 Workshops*, edited by O. Ignatenko, V. Kharchenko, V. Kobets, H. Kravtsov, Y. Tarasich, V. Ermolayev, D. Esteban, V. Yakovyna, and A. Spivakovsky, Vol. 1635 (Springer International Publishing, Cham, 2022), pp. 183–199.
[168] I. Prigogine and E. N. Hiebert, *From Being to Becoming: Time and Complexity in the Physical Sciences*, Physics Today **35**, 69 (1982).
[169] M. Costa, A. L. Goldberger, and C.-K. Peng, *Multiscale Entropy Analysis of Biological Signals*, Phys. Rev. E **71**, 021906 (2005).
[170] J. F.Donges, R. V. Donner, and J. Kurths, *Testing Time Series Irreversibility Using Complex Network Methods*, Europhysics Letters **102**, 10004 (2013).
[171] M. Zanin, A. Rodríguez-González, E. Menasalvas Ruiz, and D. Papo, *Assessing Time Series Reversibility Through Permutation Patterns*, Entropy **20**, (2018).
[172] R. Flanagan and L. Lacasa, *Irreversibility of Financial Time Series: A Graph-Theoretical Approach*, Physics Letters A **380**, 1689 (2016).
[173] A. Puglisi and D. Villamaina, *Irreversible Effects of Memory*, Europhysics Letters **88**, 30004 (2009).
[174] C. Diks, J. C. van Houwelingen, F. Takens, and J. DeGoede, *Reversibility as a Criterion for Discriminating Time Series*, Physics Letters A **201**, 221 (1995).
[175] C. S. Daw, C. E. A. Finney, and M. B. Kennel, *Symbolic Approach for Measuring Temporal "Irreversibility"*, Phys. Rev. E **62**, 1912 (2000).
[176] P. Guzik, J. Piskorski, T. Krauze, A. Wykretowicz, and H. Wysocki, *Heart Rate Asymmetry by Poincaré Plots of RR Intervals*, Biomedical Engineering / Biomedizinische Technik **51**, 272 (2006).

[177] A. Porta, S. Guzzetti, N. Montano, T. Gnecchi-Ruscone, R. Furlan, and A. Malliani, *Time Reversibility in Short-Term Heart Period Variability*, in *2006 Computers in Cardiology* (2006), pp. 77–80.
[178] L. Lacasa, A. Nuñez, É. Roldán, J. M. R. Parrondo, and B. Luque, *Time Series Irreversibility: A Visibility Graph Approach*, The European Physical Journal B **85**, (2012).
[179] A. O. Bielinskyi, S. V. Hushko, A. V. Matviychuk, O. A. Serdyuk, S. O. Semerikov, and V. N. Soloviev, *Irreversibility of Financial Time Series: A Case of Crisis*, in *Proceedings of the Selected and Revised Papers of 9th International Conference on Monitoring, Modeling & Management of Emergent Economy (M3E2-MLPEED 2021), Odessa, Ukraine, May 26-28, 2021*, edited by A. E. Kiv, V. N. Soloviev, and S. O. Semerikov, Vol. 3048 (CEUR-WS.org, 2021), pp. 134–150.
[180] A. Kiv, A. Bryukhanov, A. Bielinskyi, V. Soloviev, T. Kavetskyy, D. Dyachok, I. Donchev, and V. Lukashin, *Irreversibility of Plastic Deformation Processes in Metals*, in *Information Technology for Education, Science, and Technics*, edited by E. Faure, O. Danchenko, M. Bondarenko, Y. Tryus, C. Bazilo, and G. Zaspa (Springer Nature Switzerland, Cham, 2023), pp. 425–445.
[181] M. Costa, A. L. Goldberger, and C.-K. Peng, *Broken Asymmetry of the Human Heartbeat: Loss of Time Irreversibility in Aging and Disease*, Phys. Rev. Lett. **95**, 198102 (2005).
[182] C. L. Ehlers, J. Havstad, D. Prichard, and J. Theiler, *Low Doses of Ethanol Reduce Evidence for Nonlinear Structure in Brain Activity*, The Journal of Neuroscience **18**, 7474 (1998).
[183] C. Yan, P. Li, L. Ji, L. Yao, C. Karmakar, and C. Liu, *Area Asymmetry of Heart Rate Variability Signal*, BioMedical Engineering OnLine **16**, (2017).
[184] C. K. Karmakar, A. Khandoker, and M. Palaniswami, *Phase Asymmetry of Heart Rate Variability Signal*, Physiological Measurement **36**, 303 (2015).
[185] B. Luque, L. Lacasa, F. Ballesteros, and J. Luque, *Horizontal Visibility Graphs: Exact Results for Random Time Series*, Phys. Rev. E **80**, 046103 (2009).
[186] L. Lacasa and R. Flanagan, *Time Reversibility from Visibility Graphs of Nonstationary Processes*, Phys. Rev. E **92**, 022817 (2015).
[187] I. Grosse, P. Bernaola-Galván, P. Carpena, R. Román-Roldán, J. Oliver, and H. E. Stanley, *Analysis of Symbolic Sequences Using the Jensen-Shannon Divergence*, Physical Review E **65**, (2002).
[188] A. O. Bielinskyi, A. E. Kiv, Y. O. Prikhozha, M. A. Slusarenko, and V. N. Soloviev, *Complex Systems and Physics Education*, in *Proceedings of the 9th Workshop on Cloud Technologies in Education, CTE 2021, Kryvyi Rih, Ukraine,*

December 17, 2021, edited by A. E. Kiv, S. O. Semerikov, and M. P. Shyshkina, Vol. 3085 (CEUR-WS.org, 2021), pp. 56–80.
[189] A. A. B. Pessa and H. V. Ribeiro, *ordpy: A Python package for data analysis with permutation entropy and ordinal network methods*, Chaos: An Interdisciplinary Journal of Nonlinear Science **31**, 063110 (2021).
[190] E. P. White, B. J. Enquist, and J. L. Green, *On Estimating the Exponent of Power-Law Frequency Distributions*, Ecology **89**, 905 (2008).
[191] N. N. Taleb, *The Black Swan: Second Edition: The Impact of the Highly Improbable Fragility* (Random House Publishing Group, 2010).
[192] P. Lévy, *Calcul Des Probabilités, Par Paul lévy, ...* (Gauthier-Villars, 1925).
[193] P. Lévy, *Theorie de l'addition Des Variables Aleatoires* (Gauthier-Villars, 1954).
[194] B. V. Gnedenko and A. N. Kolmogorov, *Limit Distributions for Sums of Independent Random Variables* (Addison-Wesley, 1968).
[195] T. J. Kozubowski, M. M. Meerschaert, A. K. Panorska, and H.-P. Scheffler, *Operator Geometric Stable Laws*, Journal of Multivariate Analysis **92**, 298 (2005).
[196] A. Alvarez and P. Olivares, *Méthodes d'estimation Pour Des Lois Stables Avec Des Applications En Finance*, Journal de La Société Française de Statistique **146**, 23 (2005).
[197] J. P. Nolan, *An Algorithm for Evaluating Stable Densities in Zolotarev's (m) Parameterization*, Mathematical and Computer Modelling **29**, 229 (1999).
[198] A. Bielinskyi, V. N. Soloviev, S. Semerikov, and V. Solovieva, *Detecting Stock Crashes Using Levy Distribution*, in *Proceedings of the Selected Papers of the 8th International Conference on Monitoring, Modeling & Management of Emergent Economy, M3E2-EEMLPEED 2019, Odessa, Ukraine, May 22-24, 2019*, edited by A. Kiv, S. Semerikov, V. N. Soloviev, L. Kibalnyk, H. Danylchuk, and A. Matviychuk, Vol. 2422 (CEUR-WS.org, 2019), pp. 420–433.
[199] D. Salas-Gonzalez, J. M. Górriz, J. Ramírez, M. Schloegl, E. W. Lang, and A. Ortiz, *Parameterization of the Distribution of White and Grey Matter in MRI Using the α-Stable Distribution*, Computers in Biology and Medicine **43**, 559 (2013).
[200] V. M. Zolotarev, *One-Dimensional Stable Distributions* (American Mathematical Society, 1986).
[201] E. F. Fama and R. Roll, *Parameter Estimates for Symmetric Stable Distributions*, Journal of the American Statistical Association **66**, 331 (1971).
[202] J. H. McCulloch, *Simple Consistent Estimators of Stable Distribution Parameters*, Communications in Statistics - Simulation and Computation **15**, 1109 (1986).

[203] J. H. McCulloch, *13 Financial Applications of Stable Distributions*, in *Statistical Methods in Finance*, Vol. 14 (Elsevier, 1996), pp. 393–425.
[204] J. P. Nolan, *Maximum Likelihood Estimation and Diagnostics for Stable Distributions*, in *Lévy Processes: Theory and Applications*, edited by O. E. Barndorff-Nielsen, S. I. Resnick, and T. Mikosch (Birkhäuser Boston, Boston, MA, 2001), pp. 379–400.
[205] S. Mittnik, S. T. rachev, T. Doganoglu, and D. Chenyao, *Maximum Likelihood Estimation of Stable Paretian Models*, Mathematical and Computer Modelling **29**, 275 (1999).
[206] W. H. Dumouchel, *Stable Distributions in Statistical Inference: 1. Symmetric Stable Distributions Compared to Other Symmetric Long-Tailed Distributions*, Journal of the American Statistical Association **68**, 469 (1973).
[207] N. N. Taleb, *Statistical Consequences of Fat Tails: Real World Preasymptotics, Epistemology, and Applications*, (2022).
[208] D. J. Watts and S. H. Strogatz, *Collective Dynamics of 'Small-World' Networks*, Nature **393**, 440 (1998).
[209] A.-L. Barabási and R. Albert, *Emergence of Scaling in Random Networks*, Science **286**, 509 (1999).
[210] R. Albert and A.-L. Barabási, *Statistical Mechanics of Complex Networks*, Rev. Mod. Phys. **74**, 47 (2002).
[211] E. L. Platt, *Network Science with Python and NetworkX Quick Start Guide: Explore and Visualize Network Data Effectively* (Packt Publishing, 2019).
[212] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Fourth Edition* (MIT Press, 2022).
[213] J. Travers and S. Milgram, *An Experimental Study of the Small World Problem*, in *Social Networks*, edited by S. Leinhardt (Academic Press, 1977), pp. 179–197.
[214] A. O. Bielinskyi and V. N. Soloviev, *Complex Network Precursors of Crashes and Critical Events in the Cryptocurrency Market*, in *Proceedings of St Student Workshop on Computer Science and Software Engineering, CS and SE@SW 2018, Kryvyi Rih, Ukraine, November 30, 2018*, edited by S. O. Semerikov, A. M. Striuk, V. N. Soloviev, and A. E. Kiv, Vol. 2292 (CEUR-WS.org, 2028), pp. 37–45.
[215] A. O. Bielinskyi, V. N. Soloviev, S. V. Hushko, A. E. Kiv, and A. V. Matviychuk, *High-Order Network Analysis for Financial Crash Identification*, in *Proceedings of the Selected and Revised Papers of 10th International Conference on Monitoring, Modeling & Management of Emergent Economy (M3E2-MLPEED 2022), Virtual Event, Kryvyi Rih, Ukraine, November 17-18, 2022*, edited by H. B. Danylchuk and S. O. Semerikov, Vol. 3465 (CEUR-WS.org, 2022), pp. 132–149.

[216] A. Kiv, A. Bryukhanov, V. Soloviev, A. Bielinskyi, T. Kavetskyy, D. Dyachok, I. Donchev, and V. Lukashin, *Complex Network Methods for Plastic Deformation Dynamics in Metals*, Dynamics **3**, 34 (2023).
[217] R. V. Donner, M. Small, J. F. Donges, N. Marwan, Y. Zou, R. Xiang, and J. Kurths, *Recurrence-Based Time Series Analysis by Means of Complex Network Methods*, International Journal of Bifurcation and Chaos **21**, 1019 (2011).
[218] L. Lacasa, B. Luque, F. Ballesteros, J. Luque, and J. C. Nuño, *From Time Series to Complex Networks: The Visibility Graph*, Proceedings of the National Academy of Sciences **105**, 4972 (2008).
[219] A. O. Bielinskyi, O. A. Serdyuk, S. O. Semerikov, and V. N. Soloviev, *Econophysics of Cryptocurrency Crashes: A Systematic Review*, in *Proceedings of the Selected and Revised Papers of 9th International Conference on Monitoring, Modeling & Management of Emergent Economy (M3E2-MLPEED 2021), Odessa, Ukraine, May 26-28, 2021*, edited by A. E. Kiv, V. N. Soloviev, and S. O. Semerikov, Vol. 3048 (CEUR-WS.org, 2021), pp. 31–133.
[220] X. Lan, H. Mo, S. Chen, Q. Liu, and Y. Deng, *Fast transformation from time series to visibility graphs*, Chaos: An Interdisciplinary Journal of Nonlinear Science **25**, 083105 (2015).
[221] I. V. Bezsudnov and A. A. Snarskii, *From the Time Series to the Complex Networks: The Parametric Natural Visibility Graph*, Physica A: Statistical Mechanics and Its Applications **414**, 53 (2014).
[222] T. T. Zhou, N. D. Jin, Z. K. Gao, and Y. B. Luo, *Limited Penetrable Visibility Graph for Establishing Complex Network from Time Series*, Acta Physica Sinica **61**, 2012-3-030506 (2012).
[223] Q. Xuan, J. Zhou, K. Qiu, D. Xu, S. Zheng, and X. Yang, *CLPVG: Circular limited penetrable visibility graph as a new network model for time series*, Chaos: An Interdisciplinary Journal of Nonlinear Science **32**, 013130 (2022).
[224] F. R. K. Chung, *Spectral Graph Theory* (American Mathematical Society, 1997).
[225] N. Biggs, *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics 92)*, Bulletin of the London Mathematical Society **30**, 197 (1998).
[226] S. Butler, *Interlacing for Weighted Graphs Using the Normalized Laplacian*, Electronic Journal of Linear Algebra **16**, 90 (2007).
[227] G. Bounova and O. de Weck, *Overview of Metrics and Their Correlation Patterns for Multiple-Metric Topology Analysis on Heterogeneous Graph Ensembles*, Phys. Rev. E **85**, 016117 (2012).
[228] I. Gutman, *The Energy of a Graph*, (1978).

[229] D. M. Cvetkovic, M. Doob, and H. Sachs, *Spectra of Graphs. Theory and Application*, (1980).
[230] J. Wu, M. Barahona, Y.-J. Tan, and H.-Z. Deng, *Spectral Measure of Structural Robustness in Complex Networks*, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **41**, 1244 (2011).
[231] W. Jun, M. Barahona, T. Yue-Jin, and D. Hong-Zhong, *Natural Connectivity of Complex Networks*, Chinese Physics Letters **27**, 078902 (2010).
[232] E. Estrada, *Spectral Scaling and Good Expansion Properties in Complex Networks*, Europhysics Letters **73**, 649 (2006).
[233] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences* (Society for Industrial; Applied Mathematics, 1994).
[234] I. J. Schoenberg, *Publications of Edmund Landau*, in *Number Theory and Analysis: A Collection of Papers in Honor of Edmund Landau (1877–1938)*, edited by P. Turán (Springer US, Boston, MA, 1969), pp. 335–355.
[235] T. H. Wei, *The Algebraic Foundations of Ranking Theory* (University of Cambridge, 1952).
[236] M. G. Kendall, *Further Contributions to the Theory of Paired Comparisons*, Biometrics **11**, 43 (1955).
[237] C. Berge, *Théorie Des Graphes Et Ses Applications* (Dunod, 1958).
[238] P. Bonacich, *Technique for Analyzing Overlapping Memberships*, Sociological Methodology **4**, 176 (1972).
[239] Wikipedia, *Arnoldi Iteration*.
[240] K. Stephenson and M. Zelen, *Rethinking Centrality: Methods and Examples*, Social Networks **11**, 1 (1989).
[241] V. Latora and M. Marchiori, *Efficient Behavior of Small-World Networks*, Phys. Rev. Lett. **87**, 198701 (2001).
[242] R. Pastor-Satorras, A. Vázquez, and A. Vespignani, *Dynamical and Correlation Properties of the Internet*, Phys. Rev. Lett. **87**, 258701 (2001).
[243] S. Maslov and K. Sneppen, *Specificity and Stability in Topology of Protein Networks*, Science **296**, 910 (2002).
[244] M. E. J. Newman, *Assortative Mixing in Networks*, Phys. Rev. Lett. **89**, 208701 (2002).
[245] A. Barrat and M. Weigt, *On the Properties of Small-World Network Models*, The European Physical Journal B-Condensed Matter and Complex Systems **13**, 547 (2000).
[246] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, *Complex Networks: Structure and Dynamics*, Physics Reports **424**, 175 (2006).

[247] P. Holme, C. R. Edling, and F. Liljeros, *Structure and Time Evolution of an Internet Dating Community*, Social Networks **26**, 155 (2004).
[248] P. Holme, F. Liljeros, C. R. Edling, and B. J. Kim, *Network Bipartivity*, Phys. Rev. E **68**, 056107 (2003).
[249] P. G. Lind, M. C. González, and H. J. Herrmann, *Cycles and Clustering in Bipartite Networks*, Phys. Rev. E **72**, 056127 (2005).
[250] P. Zhang, J. Wang, X. Li, M. Li, Z. Di, and Y. Fan, *Clustering Coefficient and Community Structure of Bipartite Networks*, Physica A: Statistical Mechanics and Its Applications **387**, 6869 (2008).

Навчальне видання

МОДЕЛЮВАННЯ СКЛАДНИХ СИСТЕМ У PYTHON

Автори

Соловйов В.М., Бєлінський А.О.

Підп. до друку 26.04.2024. Формат 60×84/16. Папір офсетний.
Гарнітура Times New Roman. Умов. друк. арк. 36,04.
Наклад 100 прим. Вид. № 05-24.

Видавець Третяков Олександр Миколайович.
Свідоцтво про внесення до Державного реєстру видавців
Серія ДК No 4862 від 11.03.2015 р.
Україна, 18001, м. Черкаси, вул. Слави, 1, к. 24.
Тел.: 067 4701314. E-mail: book_brama@ukr.net