math_2024_054

Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет математики та інформатики
(повна назва інституту/факультету)

Кафедра математичного моделювання
(повна назва кафедри)

Порівняння ARIMA та LSTM моделей часових рядів

Кваліфікаційна робота

Рівень вищої освіти - перший (бакалаврський)

Виконав:
студент 4 курсу, 407 групи
Тарасов Максим Сергійович
Керівник:
доц. Дорошенко І.В.

До захисту допущено
на засіданні кафедри
математичного моделювання
протокол № _ від _____ 2024р.
Зав. кафедри _____ проф. Черевко І.М.

Чернівці – 2024

Анотація

Дана кваліфікаційна робота присвячена порівнянню моделі авторегресії інтегрованого ковзного середнього (ARIMA) та моделі довгострокової короткочасної пам’яті (LSTM) для різних часових рядів. У рамках написання роботи, створено узагальнені функції, які можуть швидко тестувати, повторювати та оптимізувати моделі ARIMA та LSTM для заданих вхідних даних часового ряду. Моделі ARIMA та LSTM використано для прогнозування семи наборів даних.

Ключові слова: ARIMA, LSTM, часові ряди, прогнозування.

Annotation

This thesis is dedicated to comparing the Autoregressive Integrated Moving Average (ARIMA) model and the Long Short-Term Memory (LSTM) model for various time series. As part of this work, generalized functions have been created that can quickly test, iterate, and optimize ARIMA and LSTM models for given time series input data. ARIMA and LSTM models are used to forecast seven datasets.

Keywords: ARIMA, LSTM, time series, forecasting.

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

_____Тарасов М.С.
(підпис)

ЗМІСТ

ВСТУП

У сучасному світі прогнозування відіграє важливу роль у різних галузях, таких як бізнес, фінанси, логістика, медицина, біологія і хімія. Воно допомагає ефективно розв'язувати різні завдання і сприяє загальному розвитку. Нейронні мережі, зокрема, стають все більш популярними у кібербезпеці, де вони використовуються для розвідки загроз, виявлення шкідливих програм та захисту кінцевих точок.

Методи прогнозування часових рядів використовують історичні та поточні дані для передбачення майбутніх значень. Аналізуючи ці дані, можна розуміти майбутні тенденції і ефективно реагувати на них. Відмінно спроектована система прогнозування може бути корисною як у бізнесі, так і у сферах національної та кібербезпеки, де вона дозволяє правильно планувати та коригувати стратегії відповідно до прогнозованих подій.

Метою цього дослідження є створення бази для ефективного прогнозування шляхом порівняння різних методів.

Вплив аналізу часових рядів на наукові дослідження можна частково оцінити, перераховуючи різноманітні сфери, де можуть виникати важливі проблеми з аналізом часових рядів. Наприклад, багато відомих часових рядів походять з економічних даних, таких як щоденні біржові котирування або місячні дані про безробіття. Епідеміологи можуть бути зацікавлені у кількості випадків захворювань, спостережених протягом певного часу. У медицині вимірювання кров'яного тиску у часі можуть бути корисні для оцінки ефективності ліків проти гіпертонії. Функціональна магнітно-резонансна томографія може використовуватися для вивчення реакції мозку на певні стимули в різних експериментальних умовах.

Багато з найбільш інтенсивних і складних застосувань методів аналізу часових рядів стосуються проблем в фізичних та природничих науках. Це пояснює основний технічний характер мови, що використовується у аналізі часових рядів. Один з найраніших записів - це щомісячні дані про сонячні

плями, що досліджувалися Шустером (1906 р.) [1]. Сучасні дослідження можуть стосуватися, наприклад, виявлення наявності потепління за глобальними температурними вимірами або впливу рівнів забруднення на щоденну смертність в Лос-Анджелесі.

Розділ 1. Часові ряди

1.1. Основні поняття

Часовий ряд - це послідовність даних, що вимірюються у різний часовий період. Вони використовуються для аналізу та прогнозування явищ, які змінюються в часі, таких як ціни на акції, температура, продажі тощо. Часові ряди даних представляють собою послідовності значень, зібраних протягом часу. Ці дані можуть бути зібрані в різних контекстах, таких як економіка, фізика, медицина, соціологія тощо.

Характер часових рядів визначається кількома ключовими аспектами:

1. Часовий аспект: Це основна властивість часових рядів, яка відображає часовий порядок збирання даних. Кожне значення даних в часовому ряді пов'язане з конкретним моментом в часі.

2. Тренди та сезонність: Часові ряди можуть містити тренди, які вказують на загальну тенденцію зміни значень з часом, а також сезонні зміни, що повторюються через фіксований період.

3. Структура кореляції: Важливо враховувати структуру кореляції між послідовними значеннями часового ряду. Кореляція може бути додатною, від'ємною або випадковою, і вона визначає, наскільки сильно значення взаємозв'язані в часі.

4. Шум: В часових рядах часто присутній шум або випадкова складова, яка не має систематичного зв'язку з іншими змінними та є випадковою.

Розуміння природи часових рядів даних допомагає вибирати відповідні методи аналізу та моделювання для виявлення закономірностей та прогнозування майбутніх значень.

Деякі проблеми та питання, що цікавлять потенційного аналітика часових рядів, найкраще виявляються при розгляді реальних експериментальних даних, зібраних у різних предметних областях. Наведені нижче випадки ілюструють деякі типові види експериментальних часових рядів, а також деякі статистичні питання, які можуть виникнути щодо таких даних.

1. Економіка: Щоденні котирування на біржі або місячні показники безробіття. Статистичні питання можуть включати виявлення трендів у ринковій активності або аналіз взаємозв'язків між різними факторами та економічними показниками.

2. Медицина: Щоденні вимірювання артеріального тиску або кількості випадків хвороби протягом тижня. Аналіз може включати виявлення сезонних змін у здоров'ї або оцінку ефективності лікування.

3. Соціологія: Щотижневі опитування про соціальні думки та настрої. Дослідження може стосуватися виявлення тенденцій у громадській думці або виявлення впливу соціальних подій на загальний настрій суспільства.

4. Екологія: Річні дані про викиди забруднювачів у певному районі. Аналіз може включати визначення трендів у забрудненості довкілля або оцінку ефективності заходів по зменшенню забруднення.

Розгляд реальних даних дозволяє аналітику краще зрозуміти специфіку проблем та вибрати належні методи аналізу для їх вирішення.

Основні етапи дослідження часових рядів включають:

1. Візуалізація даних:

- Побудова графіків для візуального аналізу даних, включаючи залежності від часу, тренди та сезонні паттерни.

2. Виявлення структурних компонентів:

- Визначення наявності та характеру трендів, сезонності та шуму у даних.

3. Оцінка стаціонарності:

- Використання статистичних тестів для оцінки стаціонарності даних. Стаціонарний часовий ряд має сталий середній та дисперсію в часі.

4. Моделювання:

- Вибір та застосування відповідних моделей для аналізу та прогнозування часового ряду, таких як ARIMA, LSTM або інші.

5. Перевірка моделі:

- Оцінка ефективності моделі за допомогою різних метрик, таких як середня квадратична помилка (MSE), коефіцієнт детермінації (R^2) тощо.

6. Прогнозування:

- Використання побудованої моделі для прогнозування майбутніх значень часового ряду.

7. Валідація:

- Перевірка точності прогнозів за допомогою порівняння прогнозованих значень з реальними даними.

Приклади досліджень часових рядів можуть включати аналіз фінансових ринків, прогнозування продажів, аналіз кліматичних змін тощо. Кожен з цих етапів вимагає уважного аналізу та методів, що відповідають конкретним властивостям даних і меті дослідження.

1.2. Статистичні моделі часових рядів

Основною метою аналізу часових рядів є розробка математичних моделей, які надають правдоподібні описи для вибіркових даних, подібних до тих, що зустрічалися у попередньому розділі. Для того, щоб надати статистичний контекст для опису характеру даних, які, здавалося б, коливаються випадковим чином з часом, ми припускаємо, що часовий ряд може бути визначений як збірка випадкових величин, індексованих в порядку їх отримання у часі. Наприклад, ми можемо розглядати часовий ряд як послідовність випадкових величин, x_1, x_2, x_3, ..., де випадкова величина x_1 позначає значення, яке отримується рядом у перший момент часу, величина x_2 позначає значення для другого періоду часу, x_3 позначає значення для третього періоду часу тощо. Загалом, збірка випадкових величин, $\{x_t\}$, індексованих по t, відома як стохастичний процес. Тут t зазвичай є дискретним і змінюється по цілих числах $t = 0, \pm1, \pm2, ...$, або деякому підмножині цілих чисел. Спостережені значення стохастичного процесу відомі як реалізація стохастичного процесу. Оскільки буде очевидно з контексту наших обговорень, ми використовуємо термін "часовий ряд",

незалежно від того, чи ми загально звертаємося до процесу, чи до конкретної реалізації, і не робимо різниці у позначеннях між цими двома концепціями.

Звичайно демонструвати вибірковий часовий ряд графічно, розташовуючи значення випадкових величин на вертикальній осі, або ординаті, з часовою шкалою на абсцисі. Зазвичай зручно з'єднувати значення на сусідніх часових періодах, щоб візуально відтворити якесь початкове гіпотетичне неперервне часовий ряд, який міг би видати ці значення як дискретний вибірковий ряд. Багато з рядів, які обговорювалися в попередньому розділі, наприклад, могли бути спостережені в будь-який момент неперервного часу і концептуально більш правильно розглядаються як неперервні часові ряди. Приближення цих рядів дискретними часовими параметрами, взятими на рівномірно розташованих точках у часі, є просто визнанням того, що вибіркові дані в основному будуть дискретними через обмеження, властиві методу збору. Крім того, аналітичні техніки стають можливими за допомогою комп'ютерів, які обмежені цифровими обчисленнями. Теоретичні розробки також базуються на ідеї того, що неперервний параметричний часовий ряд повинен бути визначений за допомогою функцій розподілу скінченного розміру, визначених на скінченній кількості точок у часі. Це не означає, що вибір інтервалу чи частоти вибірки не є надзвичайно важливою увагою. Зовнішній вигляд даних може бути повністю змінений за рахунок недостатньої частоти вибірки. Ми всі бачили, як колеса вагонеток у фільмах обертаються назад через недостатню кількість кадрів, знятих камерою. Цей феномен призводить до спотворення, яке називається аліасінгом.

Основною візуальною особливістю, яка відрізняє різні ряди, показані у прикладах.

Приклад 1.1. Білий шум

Простим видом згенерованого ряду може бути набір некорельованих випадкових величин w_t, з середнім значенням 0 та скінченною дисперсією. Часовий ряд, згенерований з некорельованих величин, використовується як

модель шуму в інженерних застосуваннях, де його називають білим шумом. Назва "білий" походить від аналогії з білим світлом і вказує на те, що всі можливі періодичні коливання присутні з однаковою силою.

Іноді нам також потрібно, щоб шум був незалежними і однаково розподіленими (*iid*) випадковими величинами з середнім значенням 0 та дисперсією. Ми розрізняємо цей випадок, називаючи його білим незалежним шумом. Особливо корисним видом білого шуму є гауссівський білий шум, де w_t є незалежними нормальними випадковими величинами з середнім значенням 0 та дисперсією σ^2. На рисунку 1.1 у верхній панелі показано набір з 500 таких випадкових величин з \(σ^2 = 1), зображених у порядку їх вибору. Отриманий ряд трохи нагадує вибух, але не є достатньо гладким, щоб слугувати правдоподібною моделлю для будь-якого іншого експериментального ряду. Графік візуально відображає суміш багатьох різних видів коливань у ряді білого шуму.

Приклад 1.2. Ковзне середнє

Ми можемо замінити ряд білого шуму w_t на ковзне середнє, щоб згладити ряд. Наприклад, розглянемо заміну w_t у Прикладі 1.1 на середнє значення його поточного значення та його найближчих сусідів у минулому і майбутньому. Це призводить до ряду, показаного на нижній панелі Рисунка 1.1. При перегляді цього ряду видно, що він є згладженою версією першого ряду, що відображає те, що повільніші коливання стають більш виразними, а деякі з швидких коливань зникають. Ми починаємо бачити схожість з індексом Південної осциляції (SOI) або з деякими fMRI рядами

Щоб відтворити Рисунок 1.1 в R, використовуйте наступні команди. Лінійна комбінація значень у часовому ряді загалом називається фільтрованим рядом; звідси і команда filter.

```
    w = rnorm(500, 0, 1) # 500 нормальних випадкових величин N(0,1)
    v = filter(w, sides=2, rep(1/3, 3)) # ковзне середнє
    par(mfrow=c(2, 1))
    plot.ts(w, main="білий шум")
```

```
plot.ts(v, main="ковзне середнє")
```

Цей код генерує ряд з 500 нормальних випадкових величин зі середнім 0 і дисперсією 1, а потім обчислює триточкове ковзне середнє. Він будує графіки білого шуму та його ковзного середнього у двох вертикально розташованих панелях

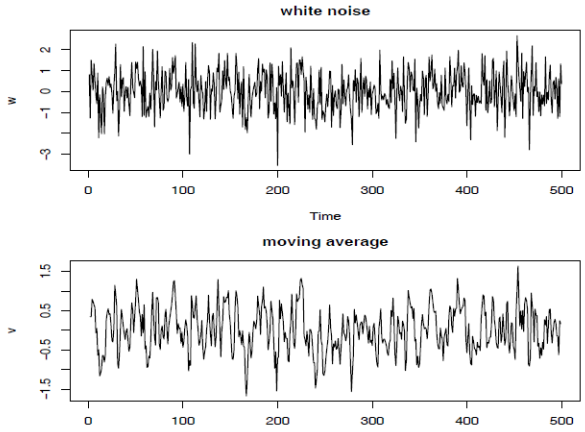


Рис. 1.1. Ряд гауссівського білого шуму (вгорі) і триточкове ковзне середнє ряду гауссівського білого шуму (внизу)

Приклад 1.3 Автокореляція

Розглянемо ряд білого шуму w_t з Прикладу 1.2 як вхідний сигнал і обчислимо вихідний сигнал, використовуючи рівняння другого порядку послідовно для \(t = 1, 2, …, 500). Рівняння представляє регресію або передбачення поточного значення x_t часового ряду як функції двох попередніх значень ряду. Таким чином, для цієї моделі використовується термін автокореляція. Існує проблема з початковими значеннями, оскільки також залежить від початкових умов. Але зараз ми припустимо, що ці значення відомі, і генеруємо наступні значення. Отриманий вихідний ряд показаний на Рисунку 1.2, і ми відзначаємо періодичну поведінку ряду, яка схожа на поведінку мовного ряду на Рисунку 1.3. Автокореляційну модель, наведена вище, та її узагальнення можна використовувати як основну модель для багатьох спостережуваних рядів.

Один зі способів моделювання та побудови графіків даних за моделлю в R — використання наступних команд (інший спосіб — використання функції `arima.sim`).

```
    w = rnorm(550, 0, 1) # 50 додаткових значень, щоб уникнути проблем з початковими умовами
    x = filter(w, filter=c(1, -0.9), method="recursive")[-(1:50)]
    plot.ts(x, main="автокореляція")
```

Цей код генерує ряд з 550 нормальних випадкових величин зі середнім 0 і дисперсією 1, щоб уникнути проблем з початковими умовами. Потім він обчислює автокореляційний ряд за допомогою рекурсивного методу з фільтром (1, -0.9) і відкидає перші 50 значень. Він будує графік автокореляційного ряду.

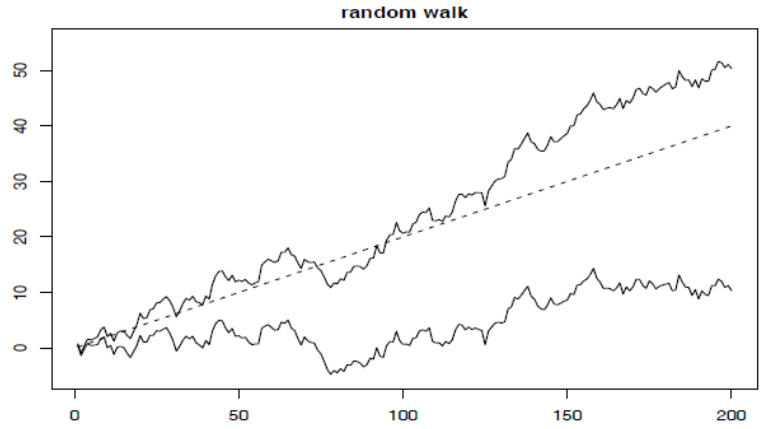


Рис. 1.2. Випадкове блукання з дрейфом

Щоб відтворити в R, скористайтеся наступним кодом:

```
    set.seed(154) # щоб ви могли відтворити результати
    w = rnorm(200, 0, 1); x = cumsum(w) # дві команди в одному рядку
    wd = w + 0.2; xd = cumsum(wd)
    plot.ts(xd, ylim=c(-5,55), main="випадкове блукання")
    lines(x)
    lines(0.2*(1:200), lty="dashed")
```

У вищенаведених прикладах ми намагалися обґрунтувати використання різних комбінацій випадкових величин, що емулюють реальні дані часових рядів. Характеристики плавності спостережуваних часових рядів були введені шляхом поєднання випадкових величин різними способами. Усереднення незалежних випадкових величин на сусідніх часових точках, як у Прикладі 1.2, або аналіз виходу різницевих рівнянь, що реагують на вхідні дані білого шуму, як у Прикладі 1.3, - це загальноприйняті

способи генерації корельованих даних. Як це зазвичай буває в статистиці, повний опис включає багатовимірну розподільну функцію спільно вибраних значень, тоді як більш економічні описи можна отримати в термінах середнього і автокореляційних функцій. Оскільки кореляція є важливою характеристикою аналізу часових рядів, найбільш корисні описові міри виражаються в термінах коваріаційних і кореляційних функцій.

1.3. Стаціонарні часові ряди

Стаціонарні часові ряди - це клас часових рядів у аналізі даних, в яких статистичні властивості ряду залишаються незмінними з плином часу. Для стаціонарного часового ряду середнє значення, дисперсія і автоковаріаційна функція (ACF) залишаються постійними протягом всього часового інтервалу.

Основні властивості стаціонарних часових рядів включають:

1. Постійне середнє значення: Середнє значення часового ряду залишається постійним з плином часу.

2. Стала дисперсія: Дисперсія ряду також залишається постійною на протязі всього часу.

3. Постійна автоковаріаційна функція (ACF): Коваріація між будь-якими двома значеннями у ряду залежить лише від часового лагу між ними і не змінюється з плином часу.

Стаціонарність є важливою для аналізу часових рядів, оскільки багато статистичних методів і моделей потребують, щоб ряди були стаціонарними для правильного прогнозування та аналізу. У випадку, якщо часовий ряд не є стаціонарним, часто потрібно виконати процедури перетворення даних, такі як диференціювання або застосування інших методів стабілізації ряду, щоб зробити його стаціонарним перед застосуванням статистичних методів аналізу або моделювання.

Розділ 2. ARIMA моделі

ARIMA моделі (авторегресивні інтегровані моделі ковзного середнього) пропонують ефективний підхід для аналізу та прогнозування часових рядів. У попередніх розділах ми розглянули функції автокореляції (ACF) та крос-кореляції (CCF), які допомагають визначити взаємозв'язки всередині та між часовими рядами на різних лагах. Ці інструменти корисні для виявлення шаблонів і залежностей у даних.

Однак класичні регресійні методи, засновані на значеннях ACF і CCF, не завжди здатні повністю відобразити всі динамічні характеристики часових рядів. Наприклад, аналіз залишків регресії часто виявляє додаткову структуру, яку ці моделі не враховують. Для вирішення цієї проблеми використовують авторегресивні (AR) та авторегресивні моделі з ковзним середнім (ARMA), які враховують кореляцію через відкладені лінійні зв'язки.

ARIMA моделі, запропоновані Боксом і Дженкінсом у 1970 році, розширюють можливості ARMA моделей, додаючи компоненти для обробки нестабільних даних. Вони об'єднують авторегресивні та ковзні середні моделі з диференціюванням для врахування трендів і сезонності. У цьому розділі ми докладно розглянемо метод Бокса-Дженкінса, який надає систематичний підхід до визначення відповідних ARIMA моделей. Також ми вивчимо методи оцінки параметрів і прогнозування з використанням цих моделей.

Автокореляційні моделі ковзного середнього (ARMA) є потужним інструментом для аналізу часових рядів. Ці моделі поєднують два компоненти: авторегресійний (AR) і ковзного середнього (MA). Такий підхід дозволяє моделювати залежність поточних значень часових рядів як від їхніх минулих значень, так і від попередніх похибок (шумів).

2.1. Авторегресійна модель (AR)

Авторегресійна модель виражає поточне значення часової серії через лінійну комбінацію попередніх значень цієї ж серії. Модель AR(p) (авторегресійна модель порядку p) має вигляд:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + \epsilon_t$$

де

x_t — поточне значення часової серії;

$\phi_1, \phi_2, \ldots, \phi_p$ — коефіцієнти моделі;

ϵ_t — білий шум (випадкова похибка) з нульовим середнім.

2.2. Модель ковзного середнього (MA)

Модель ковзного середнього виражає поточне значення часової серії через лінійну комбінацію попередніх похибок. Модель MA(q) (модель ковзного середнього порядку q) має вигляд:

$$X_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q}$$

де

x_t — поточне значення часової серії;

μ - середнє значення часового ряду,

$\epsilon_t, \epsilon_{t-1}, \epsilon_{t-2}, \ldots, \epsilon_{t-q}$ - похибки в часовому ряді на моменти часу t, t-1, t-2, ..., t-q,

$\theta_1, \theta_2, \ldots, \theta_q$ - параметри моделі, які відображають вплив кожної похибки на поточне значення часового ряду.

Зазвичай, модель ковзного середнього використовується разом з моделлю авторегресії (AR) у моделі ARMA (авторегресійна модель ковзного середнього), щоб краще описати залежності в часовому ряді та зробити більш точний прогноз

2.3. Автокореляційна модель ковзного середнього (ARMA)

Модель ARMA поєднує обидва підходи, авторегресійний і ковзного середнього, в одну модель. Модель ARMA(p, q) (автокореляційна модель ковзного середнього порядку p і q) має вигляд:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots + \phi_p x_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$$

де

x_t — поточне значення часової серії;

$\phi_1, \phi_2, \ldots, \phi_p$ — коефіцієнти авторегресійної частини моделі;

$\theta_1, \theta_2, \ldots, \theta_q$ — коефіцієнти частини ковзного середнього;

ϵ_t — білий шум (випадкова похибка) з нульовим середнім.

Застосування моделей ARMA

Моделі ARMA широко використовуються для аналізу та прогнозування часових рядів у різних галузях, таких як економіка, фінанси, метеорологія та інженерія. Основні кроки для застосування моделей ARMA включають:

1. Ідентифікація моделі: Вибір порядків p та q за допомогою аналізу автокореляційної (ACF) та часткової автокореляційної функцій (PACF).
2. Оцінка параметрів: Оцінювання коефіцієнтів моделі \(\phi\) та \(\theta\) за допомогою методів, таких як метод найменших квадратів або метод максимальної правдоподібності.
3. Діагностика моделі: Перевірка адекватності моделі через аналіз залишків (похибок), щоб переконатися, що вони ведуть себе як білий шум.
4. Прогнозування: Використання оцінених моделей для прогнозування майбутніх значень часових рядів.

Переваги та недоліки моделей ARMA

Переваги:

- Висока точність прогнозування для короткострокових прогнозів.
- Можливість моделювання складних часових структур.

Недоліки:

- Не завжди ефективні для довгострокових прогнозів.
- Моделі ARMA припускають стаціонарність рядів, що може бути обмеженням.

Моделі ARMA є основою для більш складних моделей, таких як ARIMA (авторегресійні інтегровані моделі ковзного середнього) та сезонні ARIMA (SARIMA), які використовуються для нестаціонарних часових рядів та рядів з сезонними компонентами.

2.4. Приклади

Приклад 2.1. Приклад траєкторії процесу AR(1)

Рисунок 2.1 показує часовий графік двох процесів AR(1): один з параметром ϕ=0.9 та інший з параметром ϕ=−0.9; у обох випадках дисперсія шуму σ^2 дорівнює 1. У першому випадку функція автокореляції $\rho(h)=0.9h$ для $h\geq0$, що означає позитивну кореляцію між спостереженнями, розташованими близько один до одного у часі. Це призводить до того, що спостереження, які йдуть одне за одним у часі, мають схильність бути схожими за значенням; це показано у верхній частині Рисунка 2.1 як дуже плавна траєкторія для x_t.

Тепер розглянемо випадок з параметром ϕ=−0.9, де $\rho(h)=(-0.9)$ для $h\geq0$. Тут спостереження, розташовані близько один до одного у часі, є негативно корельованими, але спостереження, розділені двома часовими пунктами, є позитивно корельованими. Це відображено у нижній частині Рисунка 2.1: наприклад, якщо спостереження x_t є позитивним, наступне спостереження x_{t+1} зазвичай буде негативним, а ще одне спостереження x_{t+2} знову буде позитивним. Таким чином, у цьому випадку траєкторія є дуже нерівною.

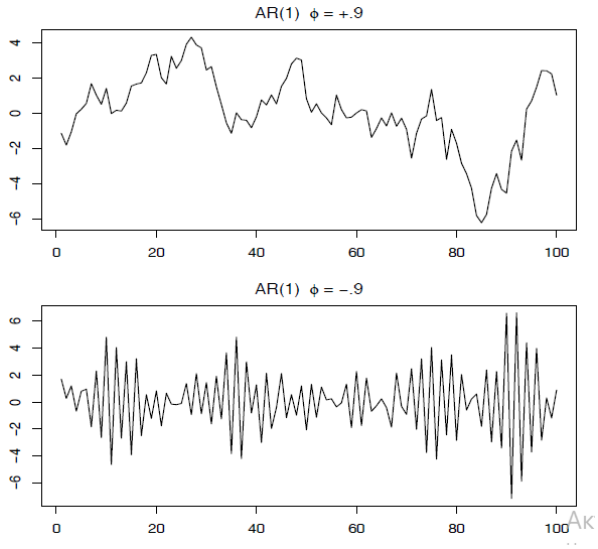


Рис. 2.1. Змодельовані AR(1) моделі

Код в R:

```
par(mfrow=c(2,1))
plot(arima.sim(list(order=c(1,0,0), ar=.9), n=100),
ylab="x",
main=(expression(AR(1)~~~phi==+.9)))
plot(arima.sim(list(order=c(1,0,0),          ar=-.9),
n=100), ylab="x",
main=(expression(AR(1)~~~phi==-.9)))
```

Приклад 2.2. Процес MA(1)
Код в R:

```
par(mfrow = c(2,1))
plot(arima.sim(list(order=c(0,0,1), ma=.5), n=100),
ylab="x",
main=(expression(MA(1)~~~theta==+.5)))
plot(arima.sim(list(order=c(0,0,1),          ma=-.5),
n=100), ylab="x",
main=(expression(MA(1)~~~theta==-.5)))
```

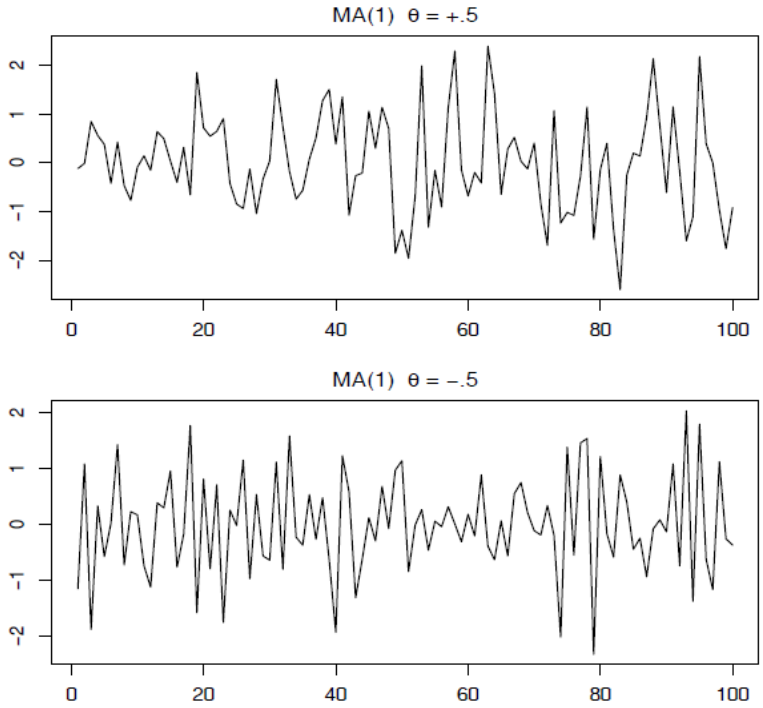


Рис. 2.2. Симульовані моделі MA(1)

Приклад 2.3. AR(2) з комплексними коренями

На рисунку 2.3 показано n = 144 спостереження з моделі AR(2):

$x_t=1.5x_{t-1}-0.75x_t-2+w_t$;

з $\sigma w_2=1$, і з обраними комплексними коренями, щоб процес проявляв псевдоциклічну поведінку із частотою один цикл на кожні 12 часових точок. Авторегресійний поліном для цієї моделі має вигляд $\phi(z)=1-1.5z+0.75z_2$. Корені $\phi(z)$ складають $1\pm i\sqrt{3}$. Для переведення кута в цикли на одиницю часу ділимо на 2π, щоб отримати 1/12 циклів на одиницю часу.

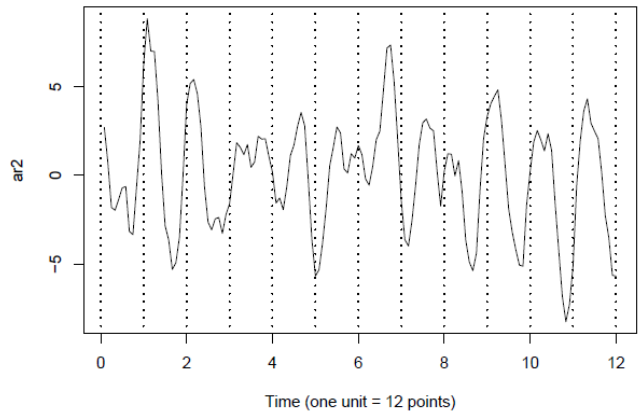


Рис. 2.3. Змодельована AR(2) модель, n = 144

Щоб розрахувати корені полінома та визначити arg у R:

```
z = c(1,-1.5,.75) # коефіцієнти полінома
(a = polyroot(z)[1]) # вивести один корінь:
1+0.57735i = 1 + i/sqrt(3)
arg = Arg(a)/(2*pi) # arg у циклах/пт
1/arg # = 12, псевдоперіод

set.seed(90210)
ar2 = arima.sim(list(order=c(2,0,0), ar=c(1.5,-
.75)), n = 144)
plot(1:144/12, ar2, type="l", xlab="Час (одиниця =
12 точок)")
abline(v=0:12, lty="dotted", lwd=2)
```

Щоб розрахувати та відобразити ACF для цієї моделі:

```
ACF = ARMAacf(ar=c(1.5,-.75), ma=0, 50)
plot(ACF, type="h", xlab="затримка")
abline(h=0)
```

2.5. Прогнозування

Прогнозування процесів ARMA (авторегресійних рухомих середніх) полягає в передбаченні майбутніх значень часового ряду на основі даних, зібраних до поточного моменту, за умови, що ряд слідує моделі ARMA. Моделі ARMA часто використовуються в аналізі часових рядів для відтворення як авторегресійних, так і рухомих складових.

Для прогнозування майбутніх значень ARMA-процесу зазвичай використовують минулі спостереження серії разом із параметрами, які були

оцінені за допомогою моделі. Процедура прогнозування включає два основні кроки:

1. Оцінка моделі: Спочатку потрібно оцінити параметри моделі ARMA на основі наявних даних. Це зазвичай включає такі техніки, як оцінка максимальної правдоподібності або метод найменших квадратів.

2. Прогнозування: Як тільки параметри оцінені, можна отримати прогнози майбутніх значень часового ряду, використовуючи оцінену модель. Це часто включає рекурсивні методи або ітераційні алгоритми, залежно від конкретної форми моделі ARMA.

Точність прогнозів залежить від кількох факторів, включаючи адекватність моделі для відтворення підтримуваних шаблонів у даних, якість оцінки параметрів та горизонт прогнозування.

На практиці програмні пакети, такі як R або statsmodels у Python, надають функції та процедури для оцінки моделей ARMA та генерації прогнозів на їх основі. Ці інструменти спрощують процес прогнозування для ARMA-процесів і дозволяють аналітикам оцінити майбутнє поведінку даних часового ряду.

Оцінка параметрів в моделях ARMA є ключовим етапом в аналізі часових рядів. Для здійснення оцінки параметрів ARMA зазвичай використовуються методи, такі як метод найменших квадратів (OLS), оцінка максимальної правдоподібності (MLE) або методи мінімальних інформаційних критеріїв (AIC, BIC).

Оцінка параметрів

Оцінка параметрів ARMA може бути складною задачею через їх взаємозалежність і можливість виникнення великої кількості параметрів. Часто використовується ітеративні методи для знаходження найкращих оцінок, такі як методи максимальної правдоподібності.

У практиці, для оцінки параметрів ARMA використовуються спеціалізовані функції та інструменти, які надаються у пакетах для статистичного аналізу, таких як R або Python. Ці інструменти дозволяють

легко виконувати оцінку параметрів ARMA за допомогою вбудованих функцій та методів, що спрощує процес аналізу часових рядів.

Приклад.[3] Запуск методу бутстреп для моделі AR(1)

Ми розглядаємо модель AR(1) з коефіцієнтом регресії, що знаходиться близько до межі причинності, та процесом помилок, який є симетричним, але не нормальним. Конкретно, розглянемо причинну модель: $x_t=\mu+\phi(x_{t-1}-\mu)+w_t$;

де μ=50, ϕ=0.95

У цьому прикладі $E(w_t)$=0 і var(w_t)=2δ^2=8. На рисунку 2.4 показано n=100 симульованих спостережень з цього процесу. Цей конкретний результат цікавий; дані виглядають так, ніби вони були згенеровані з нестаціонарного процесу з трьома різними середніми рівнями. Насправді дані були згенеровані з добре збалансованою, хоча й не нормальною, стаціонарною та причинною моделлю. Щоб показати переваги методу бутстреп, ми будемо діяти так, ніби не знаємо фактичного розподілу помилок, і продовжимо так, ніби він є нормальним; звісно, це означає, наприклад, що МНК, що базується на нормальному розподілі, для ϕ не буде реальним МНК, оскільки дані не є нормальними.

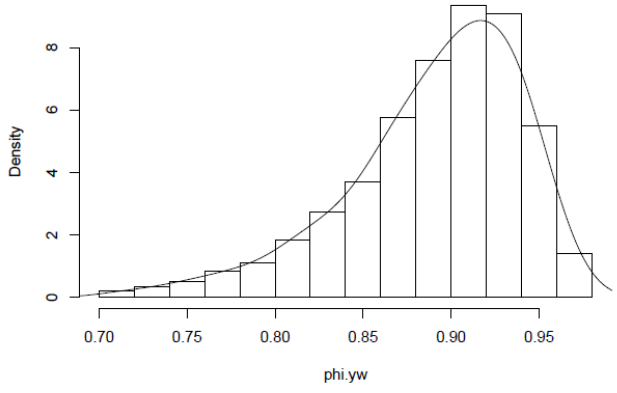


Рис. 2.4. Приблизний густинний графік оцінки Юла-Уокера для обмеженої вибірки

На основі даних, показаних на Рисунку 2.4, ми отримали оцінки Юла-Уокера.

Для оцінки скінченної вибіркової розподіленості при n = 100, ми змоделювали 1000 реалізацій цього процесу AR(1) та оцінили параметри за допомогою методу Юла-Уокера. Графік щільності скінченної вибіркової розподіленості оцінки Юла-Уокера параметра -, заснований на 1000 повторних симуляціях, показано на Рис. 2.4. Очевидно, що розподіл вибірки не дуже близький до нормального для цього обсягу вибірки. Середнє значення розподіленості становить .89, а дисперсія розподіленості складає 0,52; ці значення значно відрізняються від асимптотичних значень. Деякі квантилі скінченної вибіркової розподіленості становлять .79 (5%), .86 (25%), .90 (50%), .93 (75%) і .95 (95%).

Рядок R-коду для проведення симуляції та побудови гістограми виглядає наступним чином:

```
set.seed(111)
phi.yw <- rep(NA, 1000)
for (i in 1:1000) {
  e <- rexp(150, rate = 0.5)
  u <- runif(150, -1, 1)
  de <- e * sign(u)
  x <- 50 + arima.sim(n = 100, list(ar = 0.95),
innov = de, n.start = 50)
  phi.yw[i] <- ar.yw(x, order = 1)$ar
}
hist(phi.yw, prob = TRUE, main = "")
lines(density(phi.yw, bw = 0.015))
```

Рисунок 2.5 показує гістограму бутстрепних оцінок 200 оцінок -, отриманих з даних, показаних на Рисунку 2.3. Крім того, на Рисунку 2.5 показана оцінка щільності, заснована на гістограмі бутстрепа, а також асимптотична нормальна щільність.

Щоб провести подібну бутстрепну процедуру в R, використаємо наведені нижче команди. Важливо зазначити, що процедура оцінки в R є умовною на перше спостереження, тому перший залишок не повертається. Для обходу цієї проблеми ми просто виправляємо перше спостереження і

застосовуємо бутстреп для решти даних. Симульовані дані доступні у файлі ar1boot.

```
m <- mean(x)  # оцінка mu
fit <- ar.yw(x, order = 1)
phi <- fit$ar  # оцінка phi
nboot <- 200  # кількість бутстрепних реплікатів
resids <- fit$resid[-1]  # перше значення залишків
- NA
x.star <- x  # ініціалізуємо x*
phi.star.yw <- rep(NA, nboot)

for (i in 1:nboot) {
  resid.star <- sample(resids, replace = TRUE)
  for (t in 1:99) {
    x.star[t + 1] <- m + phi * (x.star[t] - m) +
resid.star[t]
  }
  phi.star.yw[i] <- ar.yw(x.star, order = 1)$ar
}

hist(phi.star.yw, 10, main = "", prob = TRUE, ylim
= c(0, 14), xlim = c(0.75, 1.05))
lines(density(phi.star.yw, bw = 0.02))
u <- seq(0.75, 1.05, by = 0.001)
lines(u, dnorm(u, mean = 0.96, sd = 0.03), lty =
"dashed", lwd = 2)
```

Цей код виконує бутстрепні симуляції, обчислює оцінки параметрів AR(1) за методом Юла-Вокера та відображає гістограму та оцінену щільність параметра phi.

2.6. Побудови моделей ARIMA

Для побудови моделей ARIMA існують кілька основних кроків. Ці кроки включають побудову графіка даних, можливу трансформацію даних, визначення порядку залежності моделі, оцінку параметрів, діагностику та вибір моделі. Спочатку, як і в будь-якому аналізі даних, ми повинні побудувати часовий графік даних і оглянути графік на наявність аномалій.

Якщо, наприклад, варіативність даних зростає з часом, буде необхідно трансформувати дані для стабілізації дисперсії. У таких випадках можна використовувати клас трансформацій потужності Бокса-Кокса. Крім того, конкретне застосування може вказати на відповідну трансформацію. Наприклад, якщо процес розвивається як досить маленький і стабільний процентний зміна, наприклад, як інвестиція.

Побудова моделей ARIMA передбачає декілька основних кроків. Спочатку варто проаналізувати часовий ряд та виявити можливі аномалії. Потім, якщо необхідно, можна застосувати трансформацію даних, щоб стабілізувати їх дисперсію. Наступним кроком є визначення порядку залежності моделі, що включає вибір параметрів p, d і q для моделі ARIMA(p, d, q). Після цього проводиться оцінка параметрів моделі та діагностика, щоб переконатися, що модель відповідає даним. У кінці вибирається оптимальна модель з урахуванням діагностичних результатів.

1. Аналіз даних: Починаємо з побудови графіка часового ряду та візуального огляду даних для виявлення будь-яких аномалій, трендів або сезонності.

2. Трансформація даних: Якщо у даних спостерігається нестабільність дисперсії або інші аномалії, може бути застосована трансформація даних, така як логарифмування або Box-Cox трансформація, для стабілізації дисперсії та вирішення інших проблем.

3. Визначення порядку моделі: Далі потрібно визначити порядок моделі ARIMA, тобто параметри p, d і q , де p - порядок авторегресії, d - ступінь різницювання та q - порядок ковзного середнього.

4. Параметризація моделі: Після визначення порядку моделі, застосовуємо методи оцінки параметрів для ARIMA моделі, такі як метод найменших квадратів або метод максимальної правдоподібності.

5. Діагностика моделі: Після оцінки параметрів проводиться діагностика моделі, щоб переконатися, що модель відповідає даним. Це може

включати аналіз залишкових після побудови моделі, тестування на незалежність та нормальність залишків.

6. Вибір оптимальної моделі: На основі результатів діагностики обирається найбільш оптимальна модель ARIMA для прогнозування майбутніх значень часового ряду.

Цей процес є ітеративним, і часто потребує досвіду та експертної оцінки для досягнення найкращих результатів.

Приклад. Аналіз квартального ВВП США

У цьому прикладі ми розглядаємо аналіз квартального ВВП США з 1947(1) по 2002(3), що складає 223 спостереження.

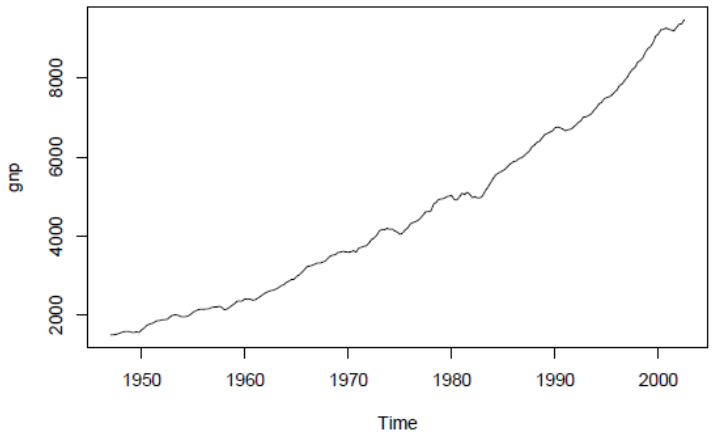


Fig. 2.6. Квартальний ВВП США з 1947(1) по 2002(3).

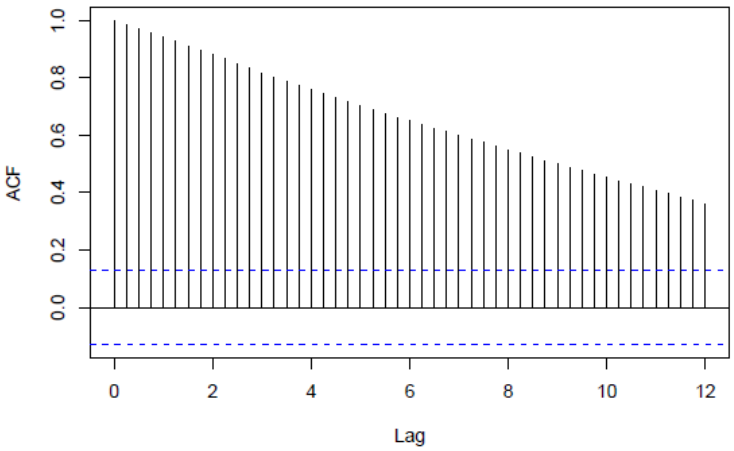


Рис 2.7. Вибіркова автокореляційна функція даних ВВП

Дані представляють реальний валовий національний продукт США в мільярдах цепних доларів 1996 року і були сезонно відкориговані. Дані були отримані з Федерального резервного банку Сент-Луїсу

(http://research.stlouisfed.org/). На рисунку 2.6 показано графік даних, скажімо, yt. Оскільки сильний тренд приховує будь-які інші ефекти, з рисунку 2.6 неясно, що дисперсія збільшується з часом. З метою демонстрації, зразок ACF даних відображений на рисунку 2.7. На рисунку 2.8 показано першу різницю даних, ryt, і тепер, коли тренд був видалений, ми можемо помітити, що варіабельність у другій половині даних більша, ніж у першій половині даних. Крім того, здається, що після віднімання залишається тренд.

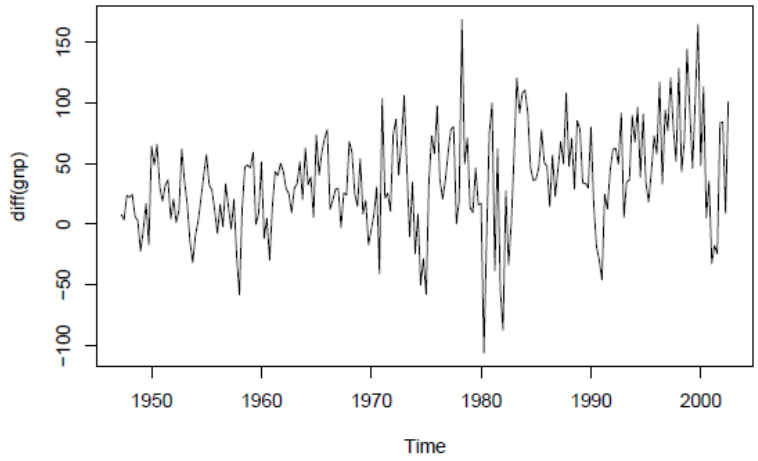


Рис. 2.8. Перша різниця даних про ВВП США

Швидкість зростання, скажімо, xt = rlog(yt), показана на рисунку 2.9 і, здається, є стабільним процесом. Крім того, ми можемо тлумачити значення xt як відсоткове щоквартальне зростання ВВП США.

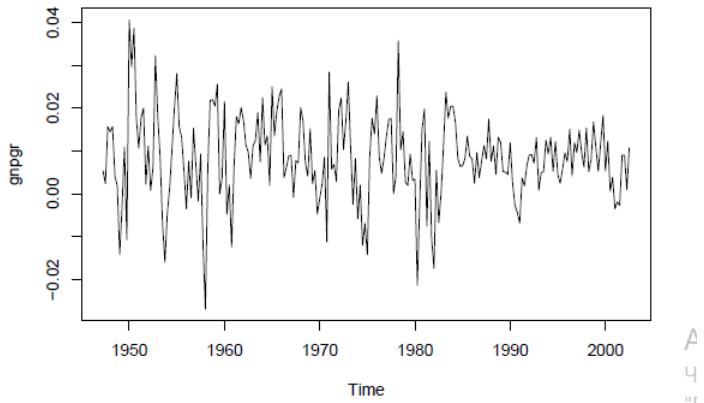


Рис. 2.9. Квартальний темп зростання ВВП США

На рисунку зображено вибіркову АКФ та часткову АКФ квартального темпу зростання. Оглядаючи вибіркову АКФ та часткову АКФ, ми можемо відчути, що вибіркова АКФ обривається на затримці 2, а часткова АКФ

поступово згасає. Це може свідчити про те, що темп зростання ВВП слідує процесу MA(2), або ж натуральний логарифм ВВП відповідає моделі ARIMA(0; 1; 2). Замість того, щоб зосереджуватися лише на одній моделі, ми також запропонуємо, що здається, що вибіркова АКФ поступово згасає, а часткова АКФ обривається на затримці 1. Це свідчить на користь моделі AR(1) для темпу зростання, або ARIMA(1; 1; 0) для натурального логарифму ВВП. Як попередній аналіз, ми розглянемо обидві моделі.

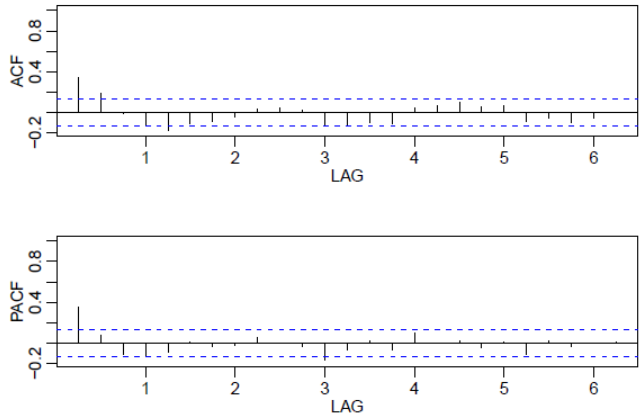


Рис. 2.10. Вибіркова АКФ та ЧАКФ квартальної ставки зростання ВВП.

Використовуючи метод максимальної правдоподібності (MLE) для підгонки моделі MA(2) до темпу зростання, x_t, отримана оцінка моделі така:

$$x_t = 0.008(0.001) + 0.303(0.065)\, b_{t-1} + 0.204(0.064)\, b_{t-2} + b_t$$

де $b_t w = -0.0094$ з урахуванням 219 ступенів свободи. Значення у дужках - відповідні оцінки стандартної похибки. Усі коефіцієнти регресії є значущими, включаючи константу. Важливо зазначити, що включення константи є обов'язковим; деякі комп'ютерні пакети за замовчуванням не враховують константу в моделі з відміненими значеннями. Однак відсутність константи може призвести до неправильних висновків про характер економіки США. Виключення константи передбачає середнє квартальне зростання нульовим, що суперечить фактичному середньому квартальному зростанню приблизно на 1% для ВВП США, як це видно на рисунку 2.10. Рекомендується додаткове дослідження для розуміння наслідків відсутності константи.

Оцінена модель AR(1) має вигляд

$$x_t=0.008(0.001)(1-0.347)+0.347(0.063)\,x_{t-1}+b_t$$

де $b_tw=-0.0095$ при 220 ступенях свободи; слід зауважити, що константа дорівнює $0.008\times(1-0.347)=0.005$.

Наступним кроком буде обговорення діагностики, але припускаючи, що обидві ці моделі підходять, як пояснити видимі відмінності оцінених моделей. Фактично, встановлені моделі майже ідентичні. Для того, щоб це показати, розглянемо модель AR(1) у вигляді, зазначеному в другому рівнянні, без константного члена; тобто

$$x_t=0.35x_{t-1}+w_t$$

Отже, отримаємо:

$$x_t\approx-0.35w_{t-1}-0.12w_{t-2}+w_t$$

Аналіз можна виконати в R за допомогою наступного коду:

```
# Побудова графіка часового ряду
plot(gnp)

# Обчислення та побудова автокореляційної функції
acf2(gnp, 50)

# Розрахунок та побудова графіка квартальної динаміки
gnpgr <- diff(log(gnp)) # Приріст
plot(gnpgr)

# Обчислення та побудова автокореляційної функції приросту
acf2(gnpgr, 24)

# Застосування функції SARIMA для моделювання AR(1)
sarima(gnpgr, 1, 0, 0)

# Застосування функції SARIMA для моделювання MA(2)
sarima(gnpgr, 0, 0, 2)

# Виведення ваг фільтру для MA(2)
```

```
ARMAtoMA(ar = 0.35, ma = 0, 10)
```

Цей код допоможе вам побудувати графіки та виконати аналіз ряду в R.

Перевірка часового графіку стандартизованих залишкових членів на Рисунку 2.11 не показує очевидних закономірностей. Варто зазначити, що є викиди, проте кілька значень перевищують 3 стандартні відхилення за магнітудою. Автокореляційна функція стандартизованих залишкових членів не показує явних відхилень від припущень моделі, і значення Q-статистики ніколи не є значущими для показаних лагів. Нормальний Q-Q графік залишкових членів виявляє відхилення від нормальності на хвостах через викиди, які переважно відбулися у 1950-х та на початку 1980-х років.

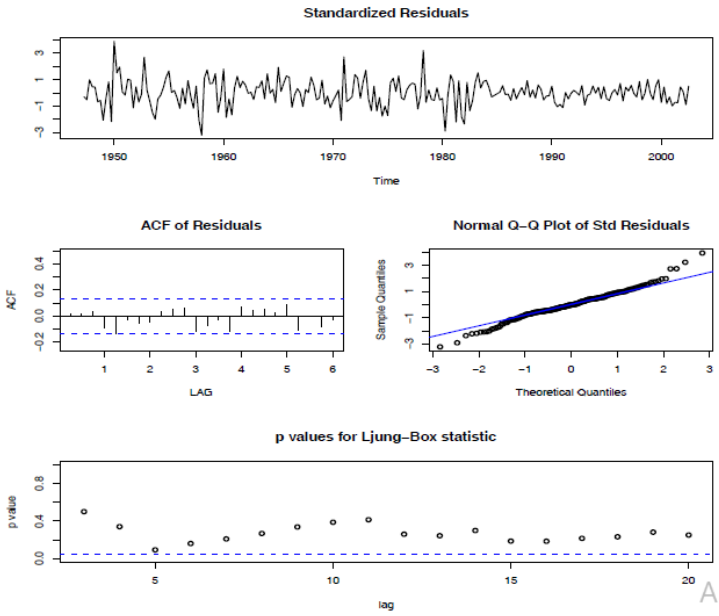


Fig. 2.11. Діагностика залишкових членів для моделі MA(2) на прирості ВВП

Модель, здається, відповідає досить добре, за винятком ситуації, коли для розподілу мають бути використані ті, що мають тяжкі хвости, ніж нормальний розподіл.

Розділ 3. Стратегії прогнозування часових рядів

Прогнозування часових рядів є важливою галуззю машинного навчання з метою передбачення майбутнього. Прогноз майбутніх точок грає вирішальну роль у прийнятті рішень в таких самих областях, як прогнозування попиту. Дані, які потрібні для прогнозування часових рядів, класифікуються на два типи: один - це часові ряди, а інший - це дані з моментами часу.

3.1. Однокрокове прогнозування

Вибір стратегії прогнозування залежить від області застосування та обсягу даних. Прогнози на один крок є актуальними у випадках, коли потрібно короткострокові передбачення. Наприклад, вимірювання кількох секунд або хвилин належать до короткострокових. Для таких ситуацій корисно розраховувати прогноз на один крок вперед. Прогноз на один крок (t+1) отримується шляхом подачі поточних та попередніх точок (t, t-1, ..., t-n) у вибрану модель (рисунок 3.1):

$$F(t+1)=M(o(t),\ldots,o(t-n),$$

де $F(t+1)$ - прогноз на час $(t+1)$, M - модель, а o(t) - значення у момент часу t [3].

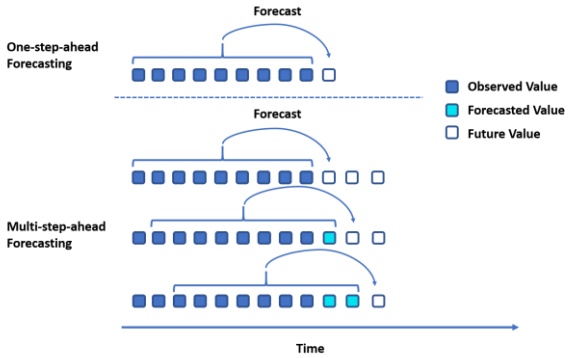


Рис.3.1. Прогнозування на один крок вперед та рекурсивне прогнозування на багато кроків вперед

3.2 Багатокрокове прогнозування

Методи багатокрокового прогнозування є важливими там, де потрібне передбачення на довгий період. Існує кілька стратегій для багатошагового прогнозування. Пряма стратегія передбачає створення окремих N прогнозних моделей для передбачення N кроків. Наприклад, для прогнозування наступних двох періодів у багатошаговому підході потрібно спочатку знайти прогнозну точку F(t + 1) за допомогою моделі. Потім інша модель використовується для прогнозування другого спостереження F(t+). Проте друге спостереження не залежить від оцінки першого (рівняння 3-4) [3]:

$$F(t+1)=M_1(o(t),\ldots,o(t-n),$$

$$F(t+2)=M_2(o(t),\ldots,o(t-n).$$

Метод багатошагового підходу можна описати таким чином:

де h - кількість спостережень, які потрібно передбачити у майбутньому, n - порядок моделі, f_h - будь-який тренер. Прямий підхід не передбачає жодних зібраних помилок, оскільки не використовує жодного передбаченого значення як дані. Однак це не гарантує жодного зв'язку між точками передбачення, оскільки всі моделі навчаються незалежно. Інша стратегія - рекурсивна, коли прогноз - це передбачення на один крок, а потім його додають до історії для здійснення наступного передбачення.

$$Y_{t+h}=f_h(y_t,\ldots,y_{t-n+1}).$$

3.3. Архітектура рекурентних нейронних мереж (РРН)

Традиційні нейронні мережі не підходять для прогнозування часових рядів, оскільки вони розглядають кожний вхід та вихід незалежно один від одного. Використання такої мережі для прогнозування часових рядів не є ефективним, оскільки ми прогнозуємо майбутні значення на основі

історичних даних. Замість цього, можна скористатися рекурентними нейронними мережами (РРН), які враховують залежності від попередніх точок даних і є більш ефективними для прогнозування часових рядів. На рисунку 3.2 зображено частину нейронної мережі, яка приймає вхідні дані x_t і видає значення h_t. Ця петля дозволяє переносити дані з одного кроку мережі на наступний, використовуючи прихований стан [4]. Однак РРН мають проблему з вивченням довгострокових залежностей. Коротко кажучи, проблема виникає через зникання градієнту, яке також відбувається в традиційних нейронних мережах. Останні шари мережі під час зворотного розповсюдження не змінюють градієнт значно. У РРН це відбувається з даними у послідовності часового ряду, оскільки перші шари петлі не навчаються добре (рисунок 3.3). Проблему можна вирішити за допомогою більш продуктивних ітерацій РРН, таких як нейронні мережі з довгостроковою та короткостроковою пам'яттю (LTSM).

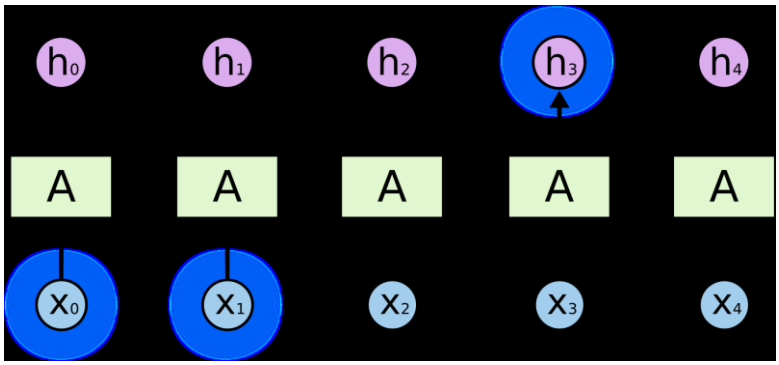


Рис.3.2. Розгорнута рекурентна нейронна мережа

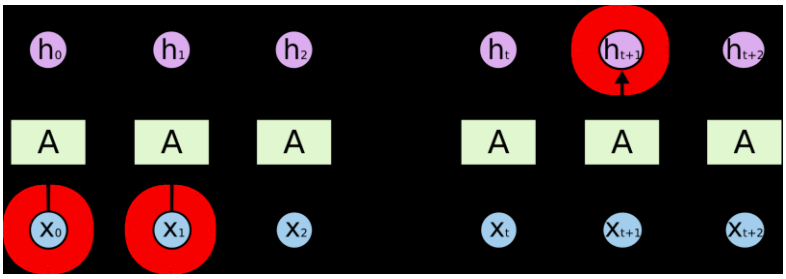


Рис.3.3. Проблема зникливого градієнта для рекурентних нейронних мереж (RNN)

3.4. Опис архітектури LSTM

Модель довготривалої та короткотривалої пам'яті (LSTM) була розроблена для подолання недоліків рекурентних нейронних мереж (RNN), таких як проблеми зникливого градієнта. У RNN складається з послідовності повторюваних модулів нейронної мережі, де кожен модуль має структуру, що включає один шар tanh. Tanh використовується для обмеження значень між -1 та 1. Вхідні дані зберігаються в прихованому стані h_t, який також містить дані з попереднього прихованого стану h_{t-1}. Це досить просто. У LSTM також є така ж послідовність повторюваних модулів, як у RNN, за винятком того, що структура модуля LSTM складається з чотирьох шарів замість одного шару, як у RNN, що допомагає вирішити проблему зникливого градієнта у RNN. На рисунку 3.4 показана різниця.

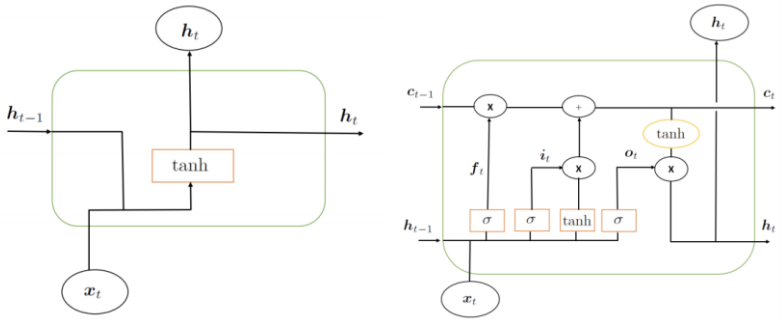


Рис.3.4. Модуль RNN (зліва) та модуль LSTM (справа)

Модель довготривалої короткотермінової пам'яті (LSTM) є варіантом рекурентної нейронної мережі (RNN), спеціально розробленим для роботи з послідовними даними, такими як часові ряди. Основна перевага LSTM полягає в тому, що вона може ефективно управляти інформацією, яка перебуває в пам'яті моделі на тривалий час, що дозволяє їй враховувати довгострокові залежності в часі.

Основні складові LSTM включають так звані "ворота", які допомагають регулювати потік інформації в мережі. Ці ворота допомагають моделі

вирішувати проблему зникнення градієнту, що часто виникає в звичайних RNN, коли вони спробують прогнозувати довгострокові залежності в часі.

Основні компоненти LSTM включають:

1. Клітинний стан (Cell State): Це основна "пам'ять" моделі LSTM, яка зберігає інформацію на тривалий термін.

2. Ворота входу (Input Gate): Визначає, яка частина нової інформації буде додана до клітинного стану.

3. Ворота забування (Forget Gate): Визначає, яка частина інформації в клітинному стані буде забута або проігнорована.

4. Ворота виходу (Output Gate): Визначає, яка частина клітинного стану буде використана для генерації виходу.

Ці компоненти дозволяють LSTM ефективно управляти інформацією в часі, що робить її дуже потужним інструментом для прогнозування часових рядів. Вони дозволяють моделі враховувати довгострокові залежності інформації, що дозволяє їй докладніше аналізувати та прогнозувати майбутні значення у порівнянні з іншими типами нейронних мереж.

Розділ 4. Порівняння моделей часових рядів ARIMA та LSTM

4.1. Основні питання дослідження

У даному розділі здійснено порівняння моделі авторегресії інтегрованого ковзного середнього (ARIMA) і моделі довгострокової короткочасної пам’яті (LSTM) для різних часових рядів. У рамках написання дипломної роботи, створено узагальнені функції, які можуть швидко тестувати, повторювати та оптимізувати моделі ARIMA та LSTM для заданих вхідних даних часового ряду. Моделі ARIMA та LSTM використано для прогнозування семи наборів даних:

- S&P 500 historical data
- "LeBron James" Google Trends
- "Coldbrew" Google Trends
- "Kentucky Derby" Google Trends
- "Gilmore Girls" Google Trends
- "Olympics" Google Trends
- "Zika Virus" Google Trends

Вказані набори даних вибрано через їх унікальні форми, щоб перевірити різні сезонні зміни, зростання значень з часом і різкі розбіжності. Для кожного набору даних побудовано графік вихідних даних та результати тестування поза вибіркою.

Крім того, у дослідження включено результати, які використовують фільтрацію Гауса для згладжування вихідного набору даних, з подальшим моделюванням згладженого набору даних. Згладжені результати моделювання порівнюються з вихідними даними часового ряду для того, щоб визначити, чи покращила фільтрація продуктивність моделей.

Отже, основні питання, на які відповідаємо у даному розділі такі:

- Яка модель показала кращий результат при прогнозуванні даних поза вибіркою: ARIMA чи LSTM?
- Чи покращує фільтрація Гауса результати моделі на вихідному невідфільтрованому наборі даних?

4.2. Моделювання ARIMA та LSTM в Python

І Statsmodels, і Keras надають потужні інструменти для моделювання часових рядів за допомогою ARIMA та LSTM відповідно. Вибір між ними залежить від характеру даних та вимог до моделі. ARIMA, реалізована в Statsmodels, підходить для аналізу стаціонарних часових рядів з автокореляцією, тоді як LSTM у Keras є кращим вибором для складних даних з довгостроковими залежностями.

4.2.1. Statsmodels для моделювання ARIMA

Statsmodels є потужним інструментом для статистичного аналізу та економетричного моделювання в Python. Він забезпечує широкий спектр методів для аналізу часових рядів, включаючи ARIMA (Автогресивна інтегрована модель ковзного середнього). ARIMA використовується для моделювання часових рядів, що демонструють автокореляцію, тобто залежність поточних значень від попередніх.

Основна ідея ARIMA полягає в поєднанні трьох компонентів: автогресії (AR), інтеграції (I) та ковзного середнього (MA). AR-компонент моделює залежність між поточним і попередніми значеннями, I-компонент застосовується для перетворення нестаціонарних рядів у стаціонарні шляхом диференціювання, а MA-компонент враховує залежність поточних значень від попередніх помилок.

За допомогою Statsmodels можна легко створити, оцінити та передбачити моделі ARIMA. Пакет надає функцію ARIMA, яка дозволяє визначити порядок моделі (p, d, q) та інші параметри. Після оцінки моделі можна використовувати її для прогнозування майбутніх значень та оцінки точності моделі. Statsmodels також забезпечує зручні інструменти для діагностики моделей та візуалізації результатів.

4.2.1. Keras для моделювання LSTM

Keras, високорівневий API для нейронних мереж, є популярним вибором для побудови та навчання рекурентних нейронних мереж (RNN), включаючи LSTM (довготривалу короткочасну пам'ять). LSTM-моделі особливо корисні для аналізу часових рядів, оскільки вони можуть вловлювати довгострокові залежності в даних.

LSTM складається з комірок, які можуть зберігати інформацію протягом тривалого часу, використовуючи механізми забування та оновлення. Це дозволяє LSTM ефективно справлятися з проблемами градієнтного зникання, з якими часто стикаються традиційні RNN.

Використання Keras для побудови LSTM-моделей є досить простим. Спочатку визначається модель, додаючи шари LSTM та інші необхідні шари, такі як Dropout або Dense. Після цього модель компілюється з вибраною функцією втрат та оптимізатором. Процес навчання здійснюється шляхом подачі навчальних даних та налаштування кількості епох та розміру пакету.

Завдяки своїй простоті та ефективності, Keras дозволяє швидко створювати та експериментувати з моделями LSTM, що робить його відмінним інструментом для задач прогнозування часових рядів.

4.3. Опис вхідних даних

Для здійснення порівняння ефективності моделей ARIMA та LSTM у роботі розглянуто 7 наборів даних:

- S&P 500 Historical Data

Дані про історичні значення індексу S&P 500 містять інформацію про цінові зміни провідного фондового індексу США, який включає 500 найбільших компаній з високою капіталізацією, що торгуються на фондових ринках. Ці дані є важливим показником стану американської економіки та використовуються для аналізу ринкових тенденцій, оцінки інвестиційних ризиків і прийняття рішень щодо управління активами.

- "LeBron James" Google Trends

Дані з Google Trends за запитом "LeBron James" відображають інтерес користувачів до одного з найвідоміших баскетболістів сучасності. Ці дані показують, як змінювалася популярність Леброна Джеймса серед користувачів інтернету в різні періоди часу, і можуть використовуватися для аналізу впливу його кар'єри, маркетингових кампаній і громадських подій на рівень пошукових запитів.

- "Coldbrew" Google Trends

Google Trends для запиту "Coldbrew" надають інформацію про динаміку інтересу до холодної кави серед користувачів інтернету. Ці дані дозволяють аналізувати сезонні тренди, вплив маркетингових акцій, популярність цього напою в різних регіонах і зміни в споживчих уподобаннях щодо холодної кави.

- "Kentucky Derby" Google Trends

Дані Google Trends за запитом "Kentucky Derby" відображають інтерес до одного з найвідоміших кінних перегонів у світі, який щорічно проходить в Кентуккі, США. Ці дані дозволяють відстежувати пік популярності цього заходу, аналізувати вплив культурних та соціальних факторів на інтерес до перегонів, а також оцінювати ефективність маркетингових кампаній.

- "Gilmore Girls" Google Trends

Google Trends для запиту "Gilmore Girls" показують динаміку інтересу до популярного телесеріалу, який має широку базу прихильників. Ці дані допомагають зрозуміти, як змінювався інтерес до серіалу протягом його трансляції та після її завершення, а також як випуск нових сезонів чи епізодів впливав на пошукову активність.

- "Olympics" Google Trends

Дані Google Trends за запитом "Olympics" відображають глобальний інтерес до Олімпійських ігор. Вони показують, як змінюється популярність цього міжнародного спортивного заходу в різні періоди, включаючи періоди перед і після ігор. Ці дані корисні для аналізу впливу Олімпійських ігор на громадськість, економіку та медіа.

- "Zika Virus" Google Trends

Google Trends для запиту "Zika Virus" надають інформацію про рівень інтересу до вірусу Зіка, особливо під час спалахів хвороби. Ці дані допомагають відстежувати, як інформаційні кампанії та спалахи захворювання впливають на пошукову активність, а також можуть бути корисними для аналізу реакції громадськості на епідеміологічні загрози.

Дані про історичні значення індексу S&P 500 можна завантажити з кількох джерел:

- Yahoo Finance https://finance.yahoo.com/
- Google Finance https://chromewebstore.google.com/
- Quandl https://data.nasdaq.com/publishers/QDL

Решта наборів даних можна отримати за допомогою Google Trends https://trends.google.com/trends/, ввівши назву набору даних у пошуковий рядок.

4.4. Модель ARIMA

Функція arima_model(series, data_split, params, future_periods, log) створює поточний прогноз моделі ARIMA для заданого вхідного часового ряду. Це дозволяє користувачеві вибирати значення p, q і d, а також вказувати, чи здійснювати log перетворення. Етапи роботи функції включають:

1) log трансформація даних;
2) створення тренувальних/тестових вибірок;
3) створення моделі ARIMA для тренувального набору;
4) прогнозування першого значення в тестовому наборі з наступним додаванням цього значення до навчального набору та ремоделювання, прогнозування наступного значення в тестовому наборі, додавання цього другого значення до навчальної вибірки і так далі;
5) зворотне перетворення даних;

6) створення графіків і генерація показників помилок.

Ця функція використовуватиметься нижче з налаштуванням параметрів p, d, q і логарифмічного перетворення, щоб мінімізувати значення RMSE для даних поза вибіркою, пов'язаних із різними часовими рядами.

4.5. Модель нейронної мережі LSTM

Обчислення LSTM використовують три різні функції для моделювання, навчання та тестування нейронної мережі LSTM. Далі здійснюється інвертування будь-яких перетворень даних, щоб побудувати графіки та проаналізувати помилки в початковому масштабі. Кожна функція описана більш детально нижче.

Функція create_dataset(data_series, look_back, split_frac, transforms) використовується для створення наборів даних, необхідних для навчання та тестування нейронних мереж LSTM. Вона приймає часовий ряд, кількість попередніх періодів, які користувач хотів би змоделювати, дроби поділу тренувань/тестів, а також те, чи потрібно виконувати диференційні або логарифмічні перетворення даних, щоб зробити їх стаціонарними. Вона також нормалізує всі дані від 0 до 1 для введення в LSTM.

Об'єкти, що підлягають навчанню, – це, по суті, періоди часу $t-n$. Отже, якщо моделювати дані за поточний місяць $(t-0)$, вхідною функцією будуть дані за попередній місяць $(t-1)$, тоді як цільове значення буде фактичним значенням для цього поточного місяця (за умови, що воно відоме). Збільшення look_back додасть додаткові попередні місяці для включення, напр. два місяці тому $(t-2)$ і три місяці тому $(t-3)$.

Функція зворотного перетворення inverse_transforms(train_predict, y_train, test_predict, y_test, data_series, train_dates, test_dates, scaler) просто скасовує будь-які перетворення, виконані під час створення набору даних. Інверсія перетворень дозволяє базувати прогнози моделі на тому ж масштабі, що й вихідний набір даних, для більш інтуїтивної інтерпретації результатів. І

функція створення моделі, і функція зворотного перетворення автоматично викликаються у функції LSTM.

Для виклику наведеної нижче моделі LSTM lstm_model(data_series, look_back, split, transforms, lstm_params) потрібен лише набір даних часового ряду data_series, кількість бажаних періодів ретроспективного аналізу look_back, поділ тренування/тесту split, чи потрібно здійснювати log перетворення чи різницю даних, а також параметри для навчання lstm_params, такі як кількість вузлів і епох. У межах функції він створює набори даних навчання та тестування – як функції, так і залежної змінної, – а потім навчає модель LSTM, після чого виконується прогнозування даних поза вибіркою. Прогнози з моделі, а також фактичні значення потім обернено перетворюються, генеруються графіки та показники помилок.

Функція gauss_compare(original_series, predictions, data_split) дозволяє порівнювати ряди прогнозованих даних, відфільтрованих за Гауссом, із вихідними рядами даних за допомогою візуалізації та звітування про RMSE.

4.6. Застосування моделей ARIMA&LSTM до згладжування даних

- **S&P 500 Historical Data**

Нижче наведено історичні дані про ціни закриття S&P 500 з 1950 року по жовтень 2017 року. Дані усереднені за місяць і представлені нижче. Ця тенденція демонструє зростання значень з часом разом із деякими різкими підвищеннями та зниженнями. Модель ARIMA працювала трохи краще, ніж модель LSTM, але покращення були здебільшого незначними. Фільтрація даних за Гауссом підвищила точність для обох моделей. Результат значень RMSE включає:

- ARIMA RMSE (без фільтра): 45,50
- LSTM RMSE (без фільтра): 46,67
- ARIMA RMSE (відфільтровано): 24,57
- LSTM RMSE (фільтрований): 24,75

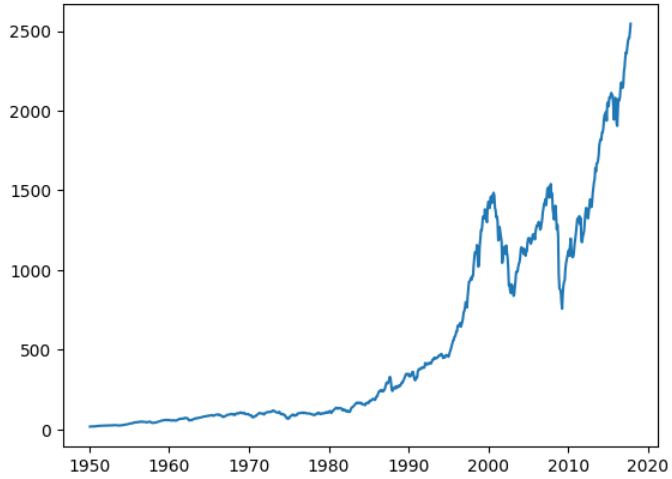
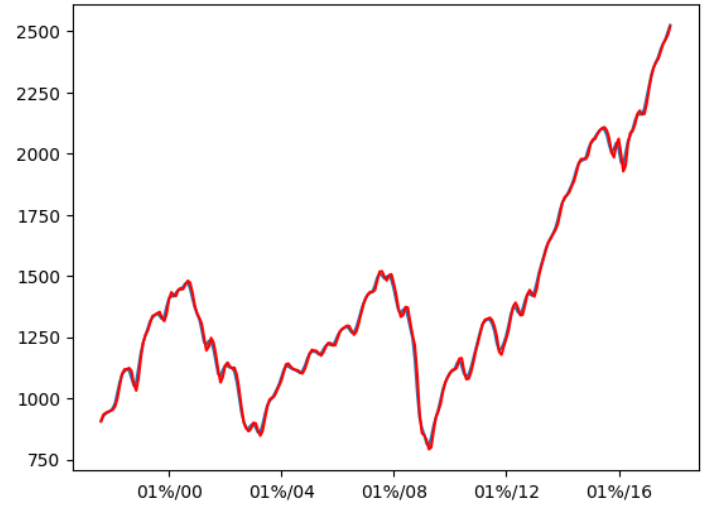


Рис. 4.1. Вхідні дані S&P 500




```
Train RMSE: 2.797
Test RMSE: 16.800
```

Рис. 4.2. LSTM згладжування даних S&P 500

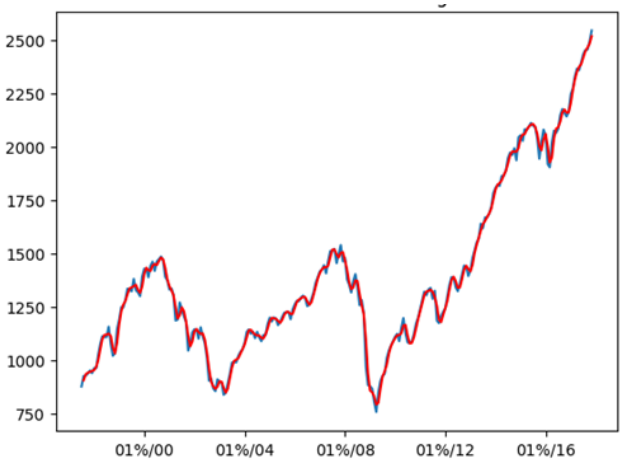


Рис. 4.3. Вхідні дані та згладжування Гаусовим фільтром

- **"LeBron James" Google Trends**

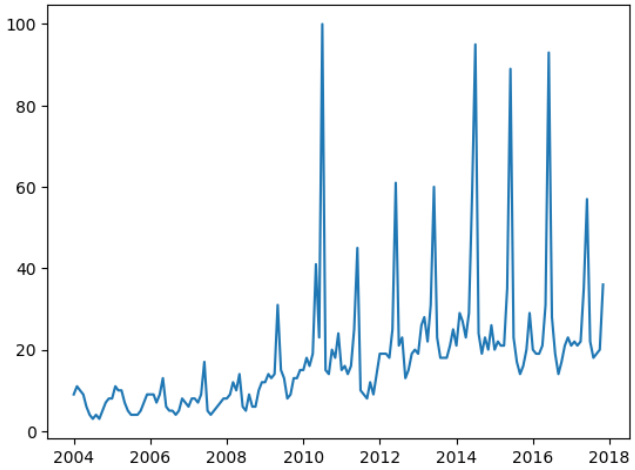


Рис. 4.4. Вхідні дані "LeBron James" Google Trends

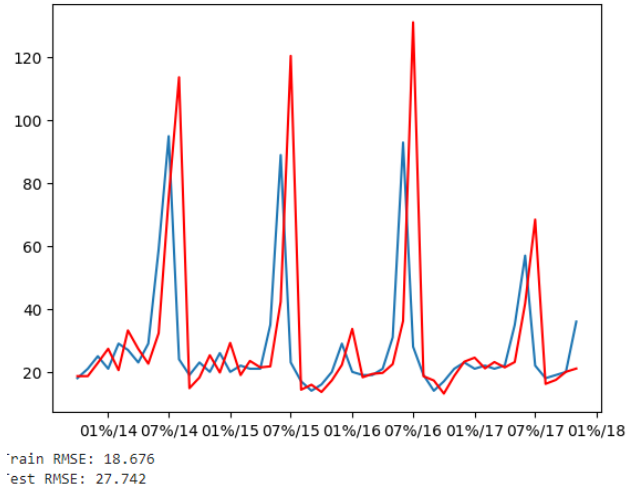


Рис. 4.5. LSTM згладжування даних "LeBron James" Google Trends

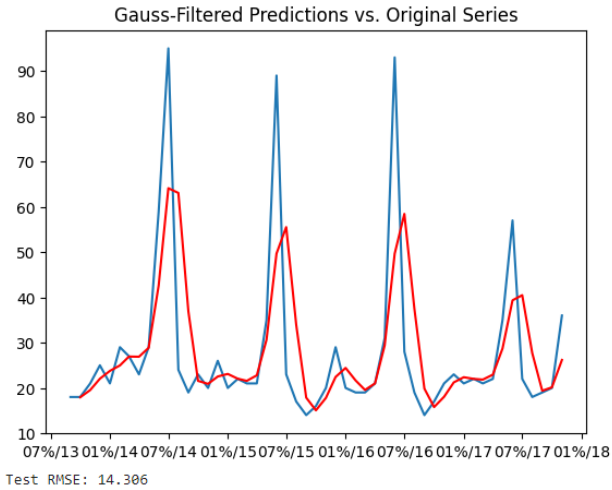


Рис. 4.6. ARIMA згладжування даних "LeBron James" Google Trends

- **"Coldbrew" Google Trends**

Далі зроблені прогнози на основі Google Trend для "Coldbrew", показаного нижче. Ця серія цікава як коливаннями (піки припадають на літні місяці), так і зростанням популярності з часом. Обидві моделі продемонстрували найкращі результати (найнижчі середньоквадратичні значення) на цьому наборі даних. Схоже, що більш плавні градієнти піків, які відбуваються протягом кількох місяців у міру потепління та похолодання, дозволяють моделям точніше прогнозувати тенденції.

Для цього набору даних модель LSTM показала кращі результати на невідфільтрованих даних, а ARIMA — на даних, відфільтрованих за Гауссом:

- ARIMA RMSE (без фільтра): 9.16
- LSTM RMSE (без фільтра): 8,60
- ARIMA RMSE (фільтрований): 3.03
- LSTM RMSE (відфільтрований): 4,54

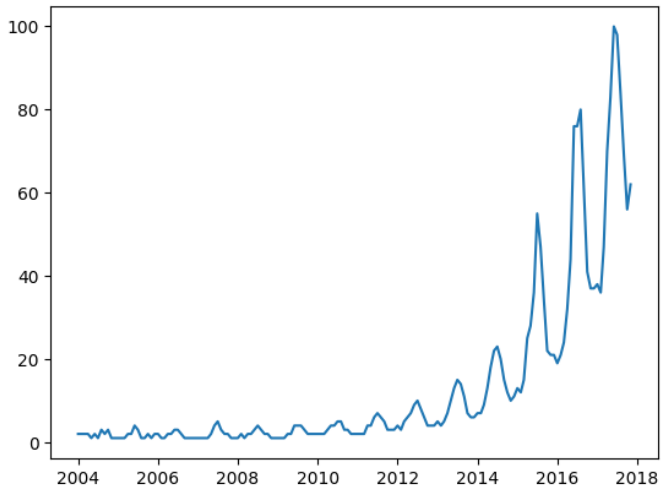


Рис. 4.7. Вхідні дані "Coldbrew" Google Trends

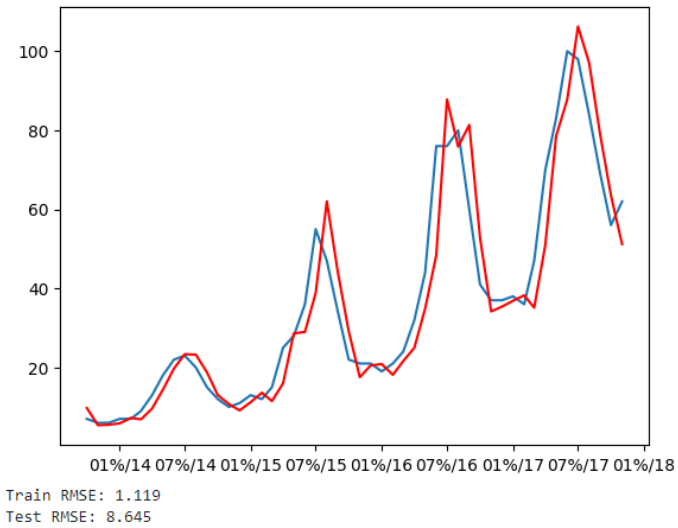


Рис. 4.8. LSTM згладжування даних "Coldbrew" Google Trends

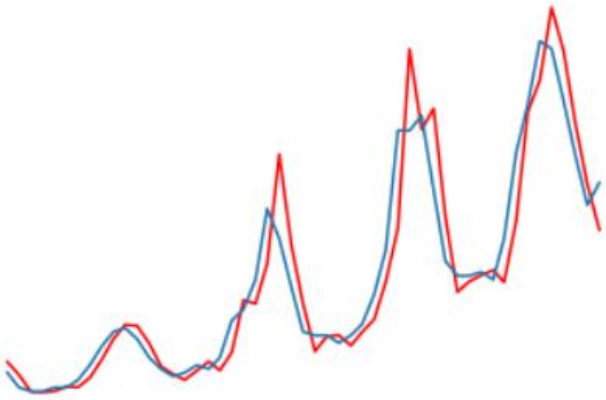


Рис 4.9. ARIMA згладжування даних "Coldbrew" Google Trends

- **Kentucky Derby Google Trend**

Кентуккі відбувається раз на рік на початку травня. Періодичну різку тенденцію навколо дати перегонів можна побачити в часовому ряді нижче. Цей часовий ряд обрано для прогнозування через його різкі градієнти, але незмінно повторювану схему.

В результаті цей набір даних мав найбільші значення RMSE з усіх змодельованих часових рядів, що означає, що, незважаючи на повторюваний шаблон, різкі градієнти все ще важко моделювати. LSTM працював так само добре або трохи краще, ніж модель ARIMA, з фільтрацією Гауса, що покращує рівень похибок на 50%. Модель ARIMA також мала деякі труднощі зі зближенням для вищих значень p і q через різкі градієнти невідфільтрованих даних.

- ARIMA RMSE (без фільтра): 30,94
- LSTM RMSE (без фільтра): 30,23
- ARIMA RMSE (відфільтрований): 15,23
- LSTM RMSE (відфільтрований): 15,23

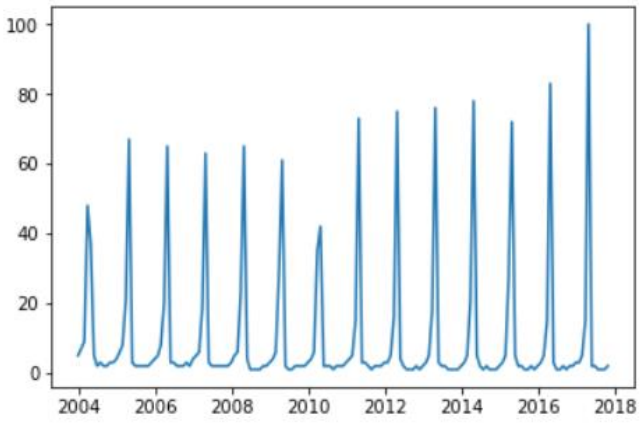


Рис 4.10. Вхідні дані Kentucky Derby Google Trend

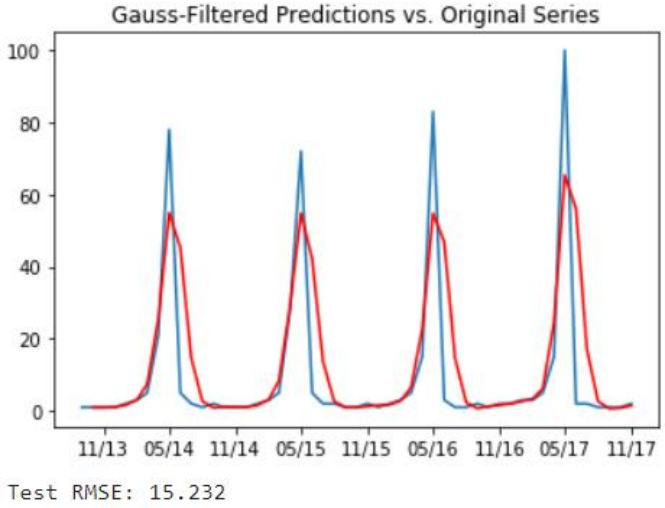


Рис 4.11. ARIMA з Гаусовим фільтром Kentucky Derby Google Trend

- **Gilmore Girls Google Trends**

Дані «Дівчата Гілмор» цікаві тим, оскільки мають цікаву різницю між навчанням і тестуванням. Перші роки демонструють велику популярність, пов’язану з шоу та його різними сезонами. Шоу закінчилося приблизно в 2007 році, але потім було повернуто Netflix для нового сезону майже через 10 років, як показує сплеск у 2017 році.

Хоча середньоквадратичні значення середньоквадратичних значень для цього ряду були другим найнижчим серед усіх досліджуваних рядів, це, ймовірно, пов'язано з тим, що існує лише один пік, для якого прогнози відповідають (або його відсутність). Багато інших серій мали кілька (3-4), що призводить до збільшення площ. Моделі ARIMA перевершила LSTM, а результати Гаусса кращі для обох моделей. Підсумкові показники помилок:

- ARIMA RMSE (без фільтра): 12,03
- LSTM RMSE (без фільтра): 13,89
- ARIMA RMSE (відфільтрований): 6,52
- LSTM RMSE (відфільтрований): 7,59

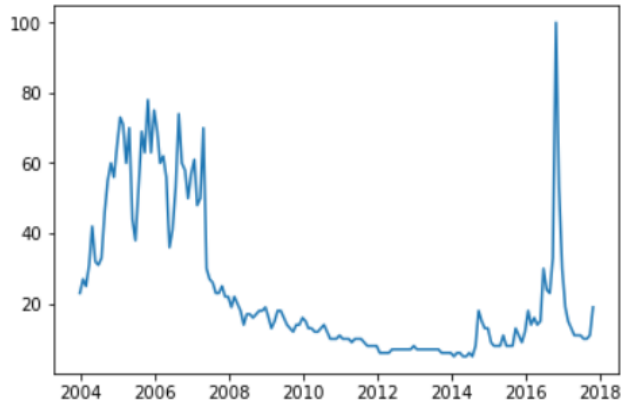


Рис 4.12. Вхідні дані Gilmore Girls Google Trends

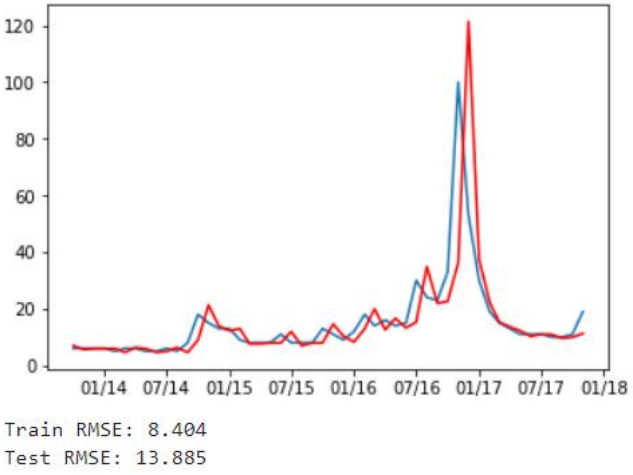


Рис 4.13. Згладжування даних Gilmore Girls Google Trends

- **Olympics Google Trend**

Ці дані цікаві через їх повторювану, але змінну структуру, що походить від пошукових запитів під час літніх Олімпійських ігор (вищі значення), за якими слідують пошуки під час зимових Олімпійських ігор. В обох випадках піки відносно вузькі з різкими градієнтами.

RMSE для даних були близькі до медіани всіх досліджуваних часових рядів, але також показали лише два піки на відміну від 4 для вищих RMSE трендів. ARIMA працює краще, ніж LSTM, як для нефільтрованих, так і для відфільтрованих наборів даних, як показано нижче.

- ARIMA RMSE (без фільтра): 17,42
- LSTM RMSE (без фільтра): 22,18
- ARIMA RMSE (відфільтрований): 9.15
- LSTM RMSE (відфільтровано): 11.08

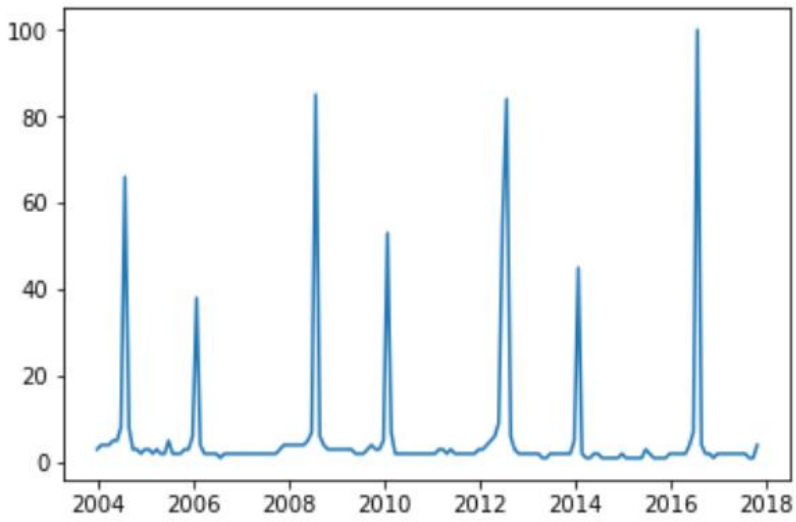


Рис 4.14. Вхідні дані Olympics Google Trend

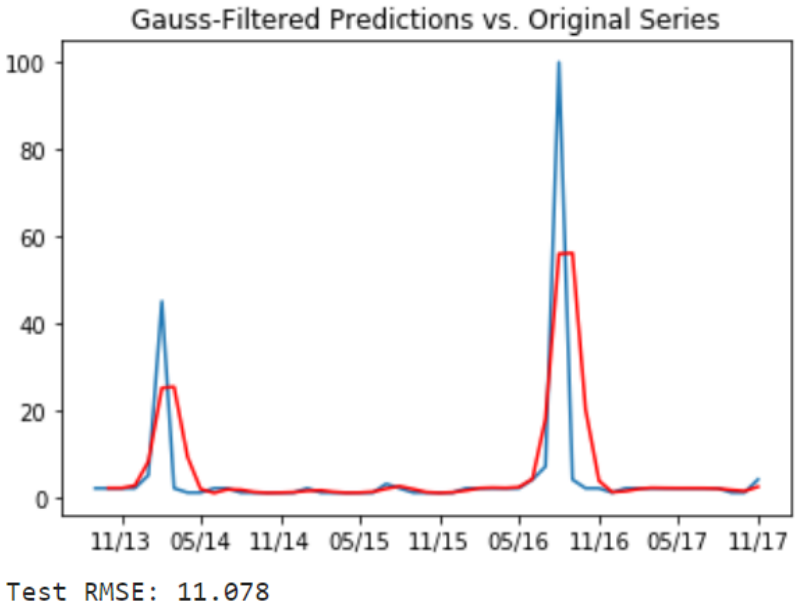


Рис 4.15. LSTM з Гауссовим фільтром для даних Olympics Google Trend

- **Zika Virus Google Trend**

Дані вірусу Зіка було обрано, щоб побачити, як алгоритми реагують на тренувальну серію з майже повною відсутністю корисної інформації.

Модель ARIMA продемонструвала трохи гірші результати з точки зору RMSE, але мала багато труднощів із наближенням. Тренувальну вибірку збільшено до 0,83, щоб зафіксувати певний рух у даних, тобто в іншому випадку отримали б одиничну похибку матриці через те, що всі значення є сталими, що призводить до марних коефіцієнтів навчання.

LSTM зміг підібрати моделі до часових , використовуючи стандартний навчальний розмір 0,7, з дещо кращими значеннями RMSE у нефільтрованому випадку.

- ARIMA RMSE (без фільтра): 16,77
- LSTM RMSE (без фільтра): 16.04
- ARIMA RMSE (відфільтрований): 8,54
- LSTM RMSE (відфільтрований): 8,59

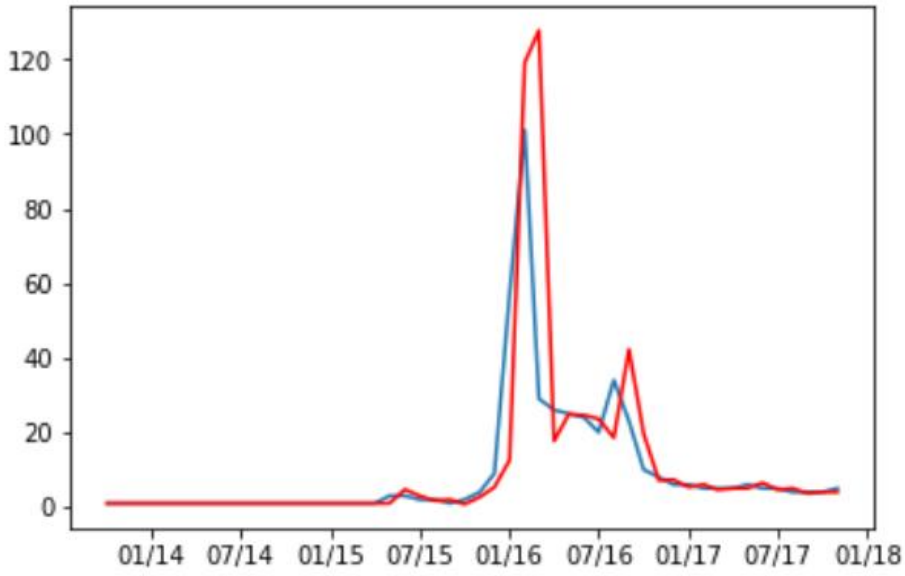


Рис 4.16. LSTM згладжування Zika Virus Google Trend

В результаті, отримано наступні результати (Таб. 1). Результати демонструють, що і ARIMA, і LSTM є якісними алгоритмами для прогнозування даних часових рядів. Загалом модель ARIMA забезпечує дещо менші похибки RMSE, але також може страждати від похибок збіжності для рядів із різкими градієнтами.

Середнє значення RMSE для моделі ARIMA без фільтру дорівнює 21,69, для моделі LSTM – 23,54. З використанням Гауссового фільтру значення RMSE для моделі ARIMA дорівнює 10,98, а для моделі LSTM – 12,22.

Таблиця 4.1. Результати дослідження

Набір даних	**RMSE**			
	Без фільтру		**З Гауссовим фільтром**	
	ARIMA	**LSTM**	**ARIMA**	**LSTM**
S&P 500 historical data	45.5	46.67	24.57	24.75
"LeBron James" Google Trends	20.01	27.17	9.82	13.74

"Coldbrew" Google Trends	9.16	8.6	3.03	4.54
"Kentucky Derby" Google Trends	30.94	30.23	15.23	15.23
"Gilmore Girls" Google Trends	12.03	13.89	6.52	7.59
"Olympics" Google Trends	17.42	22.18	9.15	11.08
"Zika Virus" Google Trends	16.77	16.04	8.54	8.59
Середні значення RMSE	**21.69**	**23.54**	**10.98**	**12.22**

ВИСНОВКИ

Модель ARIMA дала нижчі середньоквадратичні помилки (RMSE) у 5/7 досліджуваних часових рядів порівняно з моделлю LSTM. У багатьох випадках моделі давали подібні помилки, але в цілому ARIMA забезпечувала результати вищої якості, хоча їй було важко збігтися на кількох часових рядах. Результати дослідження показують, що як ARIMA, так і LSTM є ефективними алгоритмами для прогнозування часових рядів. Модель ARIMA демонструє дещо менші похибки, але може стикатися з проблемами збіжності на рядах з різкими градієнтами. Модель LSTM здатна навчатися та прогнозувати будь-які часові ряди, хоча точність прогнозів потребує окремої оцінки.

Гауссова фільтрація набору даних перед створенням моделі давала менші помилки в кожному випадку, навіть якщо порівнювати результати моделі з вихідними, невідфільтрованими даними. У середньому прогнози, відфільтровані Гауссом, зменшили RMSE майже на 100%. Додатково, фільтрація набору даних за допомогою Гауссового фільтру покращувала прогнози в усіх випадках, навіть коли порівнювати відфільтровані дані з оригінальними без фільтрації. Було б цікаво дослідити інші методи перетворення або фільтрації (наприклад, ШПФ, вейвлет-фільтри), щоб визначити можливість отримання додаткових переваг.

Середні RMSE для нефільтрованих даних, ARIMA: 21,69 та LSTM: 23,54. Середні RMSE для відфільтрованих даних за Гауссом, ARIMA: 10,98 і LSTM: 12,22.

Список використаних джерел

1. Babu, J. (2020). Multivariate Multi-step Time Series Forecasting using Stacked LSTM sequence to sequence Autoencoder. Analytics Vidhya. (https://www.analyticsvidhya.com/blog/2020/10/multivariate-multi-step-time-series-forecasting-using-stacked-lstm-sequence-to-sequence-autoencoder-in-tensorflow-2-0-keras/)

2. Brownlee, J. (2017). 4 Strategies for Multi-Step Time Series Forecasting. Machine Learning Mastery. (https://machinelearningmastery.com/multi-step-time-series-forecasting/)

3. Brownlee J. Introduction to Time Series Forecasting with Python. — 1st ed. — 2020. — 365 p.

4. Masum, S., Liu, Y., & Chiverton, J. (2018). Multi-step Time Series Forecasting of Electric Load using Machine Learning Models. School of Engineering, University of Portsmouth. (https://core.ac.uk/download/pdf/159078987.pdf)

5. Mulla, R. (2018). Hourly Energy Consumption. Kaggle. (https://www.kaggle.com/robikscube/hourly-energy-onsumption?select=PJMW_hourly.csv)

6. Sudriani Y., Ridwansyah I., Rustini H. A. Long short term memory (LSTM) recurrent neural network (RNN) for discharge level prediction and forecast in Cimandiri river, Indonesia. — 2019. — DOI: 10.10 88/1755-1315/299/1/012037.

Додаток

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import timedelta
from statsmodels.tsa.arima_model import ARIMA
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from scipy.ndimage.filters import gaussian_filter
import os
from statsmodels.tsa.arima.model import ARIMA
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'

import matplotlib.dates as mdates

def arima_model(series, data_split, params, future_periods, log):

    # # Логарифмічне перетворення даних, якщо користувач вибирає log як не true
    if log == True:
        series_dates = series.index
        series = pd.Series(np.log(series), index=series.index)

    # # створення навчальних і тестових наборів даних
    size = int(len(series) * data_split)
    train, test = series[0:size], series[size:len(series)]
    history = [val for val in train]
    predictions = []

    # # створює послідовний прогноз, тестуючи одне значення з тестового набору, а потім додає це
тестове значення
# до навчальної моделі, після чого тестує наступне тестове значення в серії
    for t in range(len(test)):
        model = ARIMA(history, order=(params[0], params[1], params[2]))
        model_fit = model.fit(disp=0)
        output = model_fit.forecast()
        yhat = output[0]
        predictions.append(yhat[0])
        obs = test[t]
        history.append(obs)

# прогнозує майбутні періоди після вхідної тестової серії
    future_forecast = model_fit.forecast(future_periods)[0]
    future_dates = [test.index[-1]+timedelta(i*365/12) for i in range(1, future_periods+1)]
    test_dates = test.index

# якщо дані спочатку були перетворені за допомогою логарифма, виконується обернене
перетворення
    if log == True:
        predictions = np.exp(predictions)
        test = pd.Series(np.exp(test), index=test_dates)
        future_forecast = np.exp(future_forecast)
```

```
# створює серії pandas з індексом datetime для передбачень та прогнозних значень
    forecast = pd.Series(future_forecast, index=future_dates)
    predictions = pd.Series(predictions, index=test_dates)

    fig = plt.figure()
    ax = fig.add_subplot(111)
    myFmt = mdates.DateFormatter('%m%/%y')
    ax.xaxis.set_major_formatter(myFmt)
    plt.plot(predictions, c='red')
    plt.plot(test)
    plt.show()

# обчислює середньоквадратичні помилки (RMSE) для передбачень за межами вибірки
    error = np.sqrt(mean_squared_error(predictions, test))
    print('Test RMSE: %.3f' % error)

    return predictions, test, future_forecast

def create_dataset(data_series, look_back, split_frac, transforms):

# логарифмує дані
    if transforms[0] == True:
        dates = data_series.index
        data_series = pd.Series(np.log(data_series), index=dates)

# виконує диференціювання даних
    if transforms[1] == True:
        dates = data_series.index
        data_series = pd.Series(data_series - data_series.shift(1), index=dates).dropna()

# масштабування значень від 0 до 1
    dates = data_series.index
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(data_series.values.reshape(-1, 1))
    data_series = pd.Series(scaled_data[:, 0], index=dates)

    df = pd.DataFrame()
    for i in range(look_back+1):
        label = ''.join(['t-', str(i)])
        df[label] = data_series.shift(i)
    df = df.dropna()
    print(df.tail())

    # розбиття даних на навчальний та тестовий набори
    size = int(split_frac*df.shape[0])
    train = df[:size]
    test = df[size:]

    X_train = train.iloc[:, 1:].values
    y_train = train.iloc[:, 0].values
    train_dates = train.index

    X_test = test.iloc[:, 1:].values
    y_test = test.iloc[:, 0].values
    test_dates = test.index
```

```
    # перетворення даних у 3 виміри для моделювання з нейронною мережею LSTM
    X_train = np.reshape(X_train, (X_train.shape[0], 1, look_back))
    X_test = np.reshape(X_test, (X_test.shape[0], 1, look_back))

    return X_train, y_train, X_test, y_test, train_dates, test_dates, scaler

def inverse_transforms(train_predict, y_train, test_predict, y_test, data_series, train_dates, test_dates,
scaler):

    train_predict = pd.Series(scaler.inverse_transform(train_predict.reshape(-1,1))[:,0], index=train_dates)
    y_train = pd.Series(scaler.inverse_transform(y_train.reshape(-1, 1))[:,0], index=train_dates)

    test_predict = pd.Series(scaler.inverse_transform(test_predict.reshape(-1, 1))[:,0], index=test_dates)
    y_test = pd.Series(scaler.inverse_transform(y_test.reshape(-1, 1))[:,0], index=test_dates)

    if (transforms[1] == True) & (transforms[0] == True):
        train_predict = pd.Series(train_predict + np.log(data_series.shift(1)), index=train_dates).dropna()
        y_train = pd.Series(y_train + np.log(data_series.shift(1)), index=train_dates).dropna()

        test_predict = pd.Series(test_predict + np.log(data_series.shift(1)), index=test_dates).dropna()
        y_test = pd.Series(y_test + np.log(data_series.shift(1)), index=test_dates).dropna()

    elif transforms[1] == True:
        train_predict = pd.Series(train_predict + data_series.shift(1), index=train_dates).dropna()
        y_train = pd.Series(y_train + data_series.shift(1), index=train_dates).dropna()

        test_predict = pd.Series(test_predict + data_series.shift(1), index=test_dates).dropna()
        y_test = pd.Series(y_test + data_series.shift(1), index=test_dates).dropna()

    if transforms[0] == True:
        train_predict = pd.Series(np.exp(train_predict), index=train_dates)
        y_train = pd.Series(np.exp(y_train), index=train_dates)

        test_predict = pd.Series(np.exp(test_predict), index=test_dates)
        y_test = pd.Series(np.exp(y_test), index=test_dates)

    return train_predict, y_train, test_predict, y_test

def lstm_model(data_series, look_back, split, transforms, lstm_params):
    np.random.seed(1)

    # створення навчальних і тестових наборів даних
    X_train, y_train, X_test, y_test, train_dates, test_dates, scaler = create_dataset(data_series, look_back,
split, transforms)

# навчання моделі
    model = Sequential()
    model.add(LSTM(lstm_params[0], input_shape=(1, look_back)))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(X_train, y_train, epochs=lstm_params[1], batch_size=1, verbose=lstm_params[2])

# прогнози
    train_predict = model.predict(X_train)
    test_predict = model.predict(X_test)

    train_predict, y_train, test_predict, y_test = \
```

```
    inverse_transforms(train_predict, y_train, test_predict, y_test, data_series, train_dates, test_dates,
scaler)

# графік прогнозів та фактичних значень
    fig = plt.figure()
    ax = fig.add_subplot(111)
    myFmt = mdates.DateFormatter('%m%/%y')
    ax.xaxis.set_major_formatter(myFmt)
    plt.plot(y_test)
    plt.plot(test_predict, color='red')
    plt.show()

# обчислення метрик RMSE
    error = np.sqrt(mean_squared_error(train_predict, y_train))
    print('Train RMSE: %.3f' % error)
    error = np.sqrt(mean_squared_error(test_predict, y_test))
    print('Test RMSE: %.3f' % error)

    return train_predict, y_train, test_predict, y_test

def gauss_compare(original_series, predictions, data_split):

# розбиття на навчальний/тестовий набір, використовуване для генерації прогнозів з гаусовим
фільтром
    size = int(len(original_series)*data_split)

    # створення графіка оригінального ряду та гаусівських фільтрованих прогнозів
    fig = plt.figure()
    ax = fig.add_subplot(111)
    myFmt = mdates.DateFormatter('%m%/%y')
    ax.xaxis.set_major_formatter(myFmt)

    plt.plot(original_series[size:])
    plt.plot(predictions, color='red')
    plt.title('Gauss-Filtered Predictions vs. Original Series')
    plt.show()

# обчислення RMSE між гаусівськими фільтрованими прогнозами та оригінальним набором даних.

    try:
        error = np.sqrt(mean_squared_error(predictions, original_series[size:]))
    except:
        error = np.sqrt(mean_squared_error(predictions, original_series[size+1:]))
    print('Test RMSE: %.3f' % error)

sp500 = pd.read_csv('sphist.csv', parse_dates=['Date'], index_col="Date")
sp500_monthly = sp500.resample('M').mean()
sp500_ts = sp500_monthly.Close
plt.plot(sp500_ts)

import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA

#  Визначення функції моделі ARIMA
def arima_model(series, data_split, params, future_periods, log):
    if log:
```

```
        series = np.log(series)

    #Розділення даних на навчальний та тестовий набори
    split_point = int(len(series) * data_split)
    train, test = series[:split_point], series[split_point:]

    # Підгон моделі ARIMA на навчальних даних
    history = list(train)
    predictions = []

    for t in range(len(test)):
        model = ARIMA(history, order=(params[0], params[1], params[2]))
        model_fit = model.fit()
        output = model_fit.forecast()
        yhat = output[0]
        predictions.append(yhat)
        history.append(test[t])

    # прогноз
    model = ARIMA(series, order=(params[0], params[1], params[2]))
    model_fit = model.fit()
    forecast = model_fit.forecast(steps=future_periods)

    if log:
        predictions = np.exp(predictions)
        test = np.exp(test)
        forecast = np.exp(forecast)

    return predictions, test, forecast

data_split = 0.7
p = 2
d = 1
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(sp500_ts, data_split, params, future_periods, log)

print("Predictions:", predictions)
print("Test data:", test)
print("Forecast:", forecast)

look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]

nodes = 4
epochs = 5
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(sp500_ts, look_back, split, transforms,
lstm_params)
```

```
sp500_ts_gauss = pd.Series(gaussian_filter(sp500_ts, sigma=1), index=sp500_ts.index).astype(float)
plt.plot(sp500_ts_gauss)
plt.show()

# запуск моделі ARIMA з гаусовим фільтром
data_split = 0.7
p = 0
d = 1
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(sp500_ts_gauss, data_split, params, future_periods, log)

# порівняння моделі ARIMA з гаусовим фільтром із оригінальною послідовністю
gauss_compare(sp500_ts, predictions, data_split)

# running LSTM with Gaussian-filtered data
look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]

nodes = 4
epochs = 10
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(sp500_ts_gauss, look_back, split, transforms,
lstm_params)

# порівняння результатів моделі з гаусовим фільтром із оригінальними даними
gauss_compare(sp500_ts, test_predict, split)

lebron = pd.Series(pd.read_csv('lebron_james.csv', header=1, parse_dates=['Month'],
index_col="Month").iloc[:,0]).astype(float)
plt.plot(lebron)

data_split = 0.7
p = 0
d = 1
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(lebron, data_split, params, future_periods, log)

look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]
```

```
nodes = 4
epochs = 10
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(lebron, look_back, split, transforms,
lstm_params)

lebron_gauss = pd.Series(gaussian_filter(lebron, sigma=1), index=lebron.index).astype(float)
plt.plot(lebron_gauss)
plt.show()

# запуск моделі ARIMA з гаусівським фільтром
data_split = 0.7
p = 2
d = 1
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(lebron_gauss, data_split, params, future_periods, log)

# порівняння моделі ARIMA з гаусівським фільтром з оригінальною серією
gauss_compare(lebron, predictions, data_split)

# running LSTM with Gaussian-filtered data
look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]

nodes = 4
epochs = 5
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(lebron_gauss, look_back, split, transforms,
lstm_params)

gauss_compare(lebron, test_predict, split)

coldbrew = pd.Series(pd.read_csv('coldbrew.csv', header=1, parse_dates=['Month'],
index_col="Month").iloc[:,0]).astype('float')
plt.plot(coldbrew)

data_split = 0.7
p = 1
d = 1
q = 0
params = [p, d, q]
future_periods = 12
log = False

predictions, test, forecast = arima_model(coldbrew, data_split, params, future_periods, log)
```

```
look_back = 1
split = 0.7
log = False
difference = True
transforms = [log, difference]

nodes = 4
epochs = 4
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(coldbrew, look_back, split, transforms,
lstm_params)

coldbrew_gauss = pd.Series(gaussian_filter(coldbrew, sigma=1), index=coldbrew.index).astype(float)
plt.plot(coldbrew_gauss)
plt.show()

# запуск моделі ARIMA з гаусівським фільтром
data_split = 0.7
p = 2
d = 1
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(coldbrew_gauss, data_split, params, future_periods, log)

gauss_compare(coldbrew, predictions, data_split)

# running LSTM with Gaussian-filtered data
look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]

nodes = 4
epochs = 20
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(coldbrew_gauss, look_back, split, transforms,
lstm_params)

gauss_compare(coldbrew, test_predict, split)

kyderby = pd.Series(pd.read_csv('kentucky_derby.csv', header=1, parse_dates=['Month'],
index_col="Month").iloc[:,0]).astype('float')
plt.plot(kyderby)

data_split = 0.7
p = 0
d = 1
q = 1
params = [p, d, q]
```

```
future_periods = 12
log = True

predictions, test, forecast = arima_model(kyderby, data_split, params, future_periods, log)

look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]

nodes = 4
epochs = 2
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(kyderby, look_back, split, transforms,
lstm_params)

kyderby_gauss = pd.Series(gaussian_filter(kyderby, sigma=1), index=kyderby.index).astype(float)
plt.plot(kyderby_gauss)
plt.show()


data_split = 0.7
p = 1
d = 1
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(kyderby_gauss, data_split, params, future_periods, log)

gauss_compare(kyderby, predictions, data_split)

# запуск LSTM з гаусівсько-фільтрованими даними
look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]

nodes = 4
epochs = 50
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(kyderby_gauss, look_back, split, transforms,
lstm_params)

gauss_compare(kyderby, test_predict, split)

gilmore_girls = pd.Series(pd.read_csv('gilmoregirls.csv', header=1, parse_dates=['Month'],
index_col="Month").iloc[:,0]).astype(float)
plt.plot(gilmore_girls)
```

```
data_split = 0.7
p = 2
d = 1
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(gilmore_girls, data_split, params, future_periods, log)

look_back = 1
split = 0.7
log = False
difference = True
transforms = [log, difference]

nodes = 4
epochs = 10
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(gilmore_girls, look_back, split, transforms,
lstm_params)

plt.plot(train_predict)
plt.plot(y_train)

gilmore_girls_gauss = pd.Series(gaussian_filter(gilmore_girls, sigma=1),
index=gilmore_girls.index).astype(float)
plt.plot(gilmore_girls_gauss)
plt.show()

data_split = 0.7
p = 0
d = 1
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(gilmore_girls_gauss, data_split, params, future_periods, log)

gauss_compare(gilmore_girls, predictions, data_split)

# запуск LSTM з гаусівсько-фільтрованими даними
look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]

nodes = 4
epochs = 20
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]
```

```
train_predict, y_train, test_predict, y_test = lstm_model(gilmore_girls_gauss, look_back, split,
transforms, lstm_params)

gauss_compare(gilmore_girls, test_predict, split)

olympics = pd.Series(pd.read_csv('olympics.csv', header=1, parse_dates=['Month'],
index_col="Month").iloc[:,0]).astype(float)
plt.plot(olympics)

data_split = 0.7
p = 2
d = 2
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(olympics, data_split, params, future_periods, log)

look_back = 1
split = 0.7
log = False
difference = True
transforms = [log, difference]

nodes = 4
epochs = 25
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(olympics, look_back, split, transforms,
lstm_params)

olympics_gauss = pd.Series(gaussian_filter(olympics, sigma=1), index=gilmore_girls.index).astype(float)
plt.plot(olympics_gauss)
plt.show()

data_split = 0.7
p = 2
d = 1
q = 0
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(olympics_gauss, data_split, params, future_periods, log)

# comparing ARIMA model with Gaussian filter to original series
gauss_compare(olympics, predictions, data_split)

look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]

nodes = 4
```

```
epochs = 20
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(olympics_gauss, look_back, split, transforms,
lstm_params)

gauss_compare(olympics, test_predict, split)

zika = pd.Series(pd.read_csv('zika.csv', header=1, parse_dates=['Month'],
index_col="Month").iloc[:,0]).astype('float')
plt.plot(zika)

data_split = 0.83
p = 0
d = 1
q = 1
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(zika+1, data_split, params, future_periods, log)

look_back = 1
split = 0.7
log = True
difference = True
transforms = [log, difference]

nodes = 4
epochs = 10
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(zika+1, look_back, split, transforms,
lstm_params)

zika_gauss = pd.Series(gaussian_filter(zika, sigma=1), index=zika.index).astype(float)
plt.plot(zika_gauss)
plt.show()

# running ARIMA model with Gaussian Filter
data_split = 0.83
p = 1
d = 2
q = 0
params = [p, d, q]
future_periods = 12
log = True

predictions, test, forecast = arima_model(zika_gauss+1, data_split, params, future_periods, log)

gauss_compare(zika, predictions, data_split)

look_back = 1
split = 0.7
log = True
```

```
difference = True
transforms = [log, difference]

nodes = 4
epochs = 20
verbose = 0 # 0=print no output, 1=most, 2=less, 3=least
lstm_params = [nodes, epochs, verbose]

train_predict, y_train, test_predict, y_test = lstm_model(zika_gauss+1, look_back, split, transforms,
lstm_params)

# порівняння результатів моделі з гаусівським фільтром з оригінальними даними
gauss_compare(zika, test_predict, split)
```