МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра прикладної математики та моделювання складних систем

Допущено до захисту
Завідувач кафедри ПМ та МСС
_____ доцент Коплик І.В.
(підпис)
«__» _____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «магістр»

спеціальність 113 «Прикладна математика»

освітньо-професійна програма «Наука про дані та моделювання складних систем»

тема роботи

«ПРОГНОЗУВАННЯ ЧАСОВИХ РЯДІВ ЗА ДОПОМОГОЮ НЕЙРОННИХ МЕРЕЖ ПРЯМОГО ПОШИРЕННЯ»

Виконавець

студентка факультету ЕлІТ

Налімова Поліна Олександрівна _____

Науковий керівник

доцент, к.ф.-м.н.

Князь Ігор Олександрович _____

Суми – 2021

РЕФЕРАТ

Кваліфікаційна робота: 42 с., 23 малюнка, 30 джерел.

Мета роботи: написання програми для прогнозування часових рядів за допомогою нейронних мереж.

Об’єкт дослідження: часовий ряд погодинного споживання електроенергії від PJM Interconnection LLC.

В роботі було досліджено класичну модель прогнозування типу SARIMA та проведено аналіз часового ряду погодинного електроспоживання.

Було написано програму для прогнозування часових рядів за допомогою нейронної мережі прямого поширення, що навчена методом зворотного розповсюдження помилки. Зроблено прогноз за допомогою моделі прогнозування SARIMA.

Порівняно результати роботи нейронної мережі з результатами роботи класичної моделі прогнозування SARIMA.

СТАЦІОНАРНІСТЬ, ЧАСОВИЙ РЯД, НЕЙРОННА МЕРЕЖА, МЕТОД ЗВОРОТНОГО РОЗПОСЮДЖЕННЯ ПОМИЛКИ.

ЗМІСТ

ВСТУП

Прогнозування, на сьогодні, це одна з найскладніших, а також найбільш затребуваних і актуальних задач аналізу даних. Основна складність прогнозування, як процесу, пов'язана з тим, що необхідно аналізувати та проводити оцінку великих обсягів даних, а також з ускладненням методів та появою нових підходів до прогнозування різних процесів. Тому, безпосередньо, стан розвитку методів прогнозування, на сьогоднішній день, залежить від розвитку інформаційних технологій.

Задача прогнозування часових рядів була актуальною і залишається такою на сьогоднішній день, оскільки передбачення є важливим, а часто і необхідним, елементом різних видів діяльності.

Часовий ряд являє собою послідовність значень певного статистичного показника, що є впорядкованим у хронологічному порядку. Прикладами часових рядів можуть бути показники технічних, природних, економічних, соціальних та інших систем.

Застосування моделі прогнозування для передбачення майбутніх значень на основі попередніх спостережень і є прогнозуванням часових рядів.

На сьогодні, прогнозування часових рядів широко застосовується в політиці, економіці та багатьох інших сферах людської діяльності. Правильні прогнози дають можливість мінімізувати ризики прийняття неправильних рішень.

Дослідження методів прогнозування ще не сказали свого останнього слова і тому є сенс розвивати цю течію з метою пошуку найкращого методу прогнозування, який буде зручним, легким, швидким і в перчу чергу точним.

Для прогнозування часового ряду необхідно виконати наступне:

- Перевірити ряд на стаціонарність
- За результатами перевірки на стаціонарність, визначити модель прогнозування та її параметри
- Визначити нейромережеву модель та сформувати навчальну вибірку на основі значень вихідного часового ряду

- Побудувати нейронну мережу прямого поширення на навчити її методом зворотного поширення помилки.

1. АНАЛІТИЧНИЙ ОГЛЯД

Сьогодні, коли інформаційні технології розвиваються так стрімко, задача прогнозування функціонального стану складних систем на основі аналізу часового ряду є дуже актуальною в багатьох різних областях людської діяльності [1, 2]. Аналіз часових рядів дає можливість виявляти (помічати) закономірності динаміки змін їх значень та оцінити значення показників, що описуються часовими рядами, в майбутньому [3].

Для рішення задач прогнозування традиційно користуються класичними моделями. Для аналізу стаціонарних часових рядів використовують: авторегресійну модель $AR(p)$, модель ковзного середнього $MA(q)$ чи модель авторегресії-ковзного середнього $ARMA(p,q)$. p та q – це параметри (порядки) моделей.

Для аналізу нестаціонарних часових рядів використовують: інтегровану модель авторегресії ковзного середнього $ARIMA(p,d,q)$ або сезонну інтегровану модель авторегресії- ковзного середнього $SARIMA(p,d,q)$ (P_s, D_s, Q_s). В даному випадку, d - це порядок різниці часового ряду, D_s – порядок різниці сезонної складової, P_s – параметр авторегресійної сезонної компоненти, Q_s – параметр сезонної складової ковзного середнього, s – величина сезонності [4].

Слід зауважити, що у звязку з періодичністю даних та їх випадковістю, класичні моделі прогнозування інколи не здатні вловити особливості даних та можуть показувати погані результати. Саме тому дослідники пропонують різні підходи та методи, які допомагають підвищити стабільність та точність прогнозування.

З цією метою в роботі [5] запропонована новітня гібридна система прогнозування, що складається з трьох модулів, таких як, модуль оптимізації, модуль прогнозування та модуль шумоподавлення.

Також, в роботі [6] викладається підхід, який за основу має методи нечіткої логіки. В даній праці, враховується, що часові ряди можуть бути адитивно розкладені на сезонну та трендову компоненти, а також можуть бути виділені

нерегулярні коливання. Тому прогнозування, в даному випадку, являє собою комбінацію індивідуального прогнозування кожної з цих складових.

Традиційні моделі прогнозування $AR(p)$, $MA(q)$, $ARMA(p,q)$, $ARIMA(p,d,q)$ та $SARIMA(p,d,q)(P_s,D_s,Q_s)$ є нелінійними функціями. Коефіцієнти в таких моделях можна визначити використавши вихідний часовий ряд. Однак нейронні мережі, що є апроксиматорами нелінійних функцій, можна використовувати, якщо замінити лінійний вид традиційних моделей на нелінійний [7, 8, 9].

Для використання нейромережевої моделі прогнозування необхідно: по-перше, визначити вид і архітектуру мережі; по-друге, параметри нейромережі з використанням навчальної вибірки, яка будується на основі значень вихідного часового ряду. Як правило, для задач прогнозування застосовують або рекурентні мережі або мережі прямого поширення [10, 11, 12, 13, 14].

Для визначення структури нейромережі досить часто застосовують генетичний алгоритм [15]. Цей же алгоритм також може використовуватися для формування складу вхідних змінних [16], але даний підхід потребує значних вирахувальних затрат.

Більш доцільним, для визначення складу вхідних змінних нейромережі, буде використання методик, які застосовуються при побудові традиційної лінійної прогнозної моделі, а саме: автокореляційні функції (АКФ) та часткові автокореляційні функції (ЧАКФ) [17].

Нестаціонарний часовий ряд, за допомогою нескладних перетворень, приводиться до стаціонарного часового ряду. Одним з таких методів є, наприклад, взяття кінцевих різниць. На основі аналізу АКФ та ЧАКФ визначаються параметри моделі $ARIMA(p,d,q)$ або моделі $SARIMA(p,d,q)(P_s,D_s,Q_s)$ відповідно при відсутності або наявності сезонної компоненти.

Отримані значення параметрів (порядків) класичних моделей часового ряду визначають кількість минулих значень часового ряду, які будуть використані в нейромережевій моделі.

2. АНАЛІЗ ЧАСОВИХ РЯДІВ

2.1. Постановка задачі

Класичні моделі прогнозування $ARIMA(p, d, q)$ та $SARIMA(p, d, q)(P_s, D_s, Q_s)$, як правило дають результати високої точності, коли масив даних є впорядкованим та повним. В іншому випадку, тобто в випадку, коли дані містять часові пропуски, такі моделі або перестають працювати або збільшується похибка. Саме тому розробляємо нейромережеву модель.

В ході роботи необхідно виконати наступні кроки:

Нехай задано значення ряду $Y = \{y_1,\ y_2, \dots, y_n\}$.

1) Провести аналіз часового ряду;
2) Визначити вид моделі прогнозування;
3) Знайти параметри моделі (для стаціонарного чи нестаціонарного ряду);
4) Визначити тип прогнозної нейромережевої моделі;
5) Сформувати навчальну вибірку, відповідно до виду прогнозної нейромережевої моделі, на основі значень вихідного часового ряду;
6) Побудувати нейронну мережу прямого поширення навчену методом зворотного розповсюдження помилки;
7) Порівняти результати нейронної мережі з результатами відповідної класичної моделі прогнозування.

2.2. Метод побудови прогнозної нейромережевої моделі часового ряду

В роботі було застосовано наступний метод побудови прогнозної нейромережевої моделі часового ряду:

1. Декомпозиція часового ряду, виділення тренду, сезонності, та шуму.
2. Перевірка ряду на стаціонарність.
3. У випадку, якщо часовий ряд є стаціонарним, тобто тренд та сезонність відсутні, на основі аналізу графіків автокореляційної функції (АКФ) та часткової автокореляційної функції (ЧАКФ) визначити параметри p та q.
4. У випадку якщо ряд є нестаціонарним, тобто наявний тренд, але відсутня сезонність, часовий ряд необхідно привести до стаціонарного виду методом кінцевих різниць. Порядок різниці d вважається дійсним тоді, коли після взяття різниці порядку d вихідний нестаціонарний ряд перетворюється на стаціонарний. Параметри p і q визначаються з графіків АКФ та ЧАКФ стаціонарного ряду.
5. У випадку якщо часовий ряд є нестаціонарним, має тренд та сезонність, його необхідно перетворити до стаціонарного виду шляхом взяття кінцевих різниць як для поточних значень ряду, так і для його сезонних складових. Порядок різниці d і порядок різниці сезонної складової D_s вважаються визначеними у тому випадку, коли при взятті різниці порядку D, а потім порядку d, вихідний нестаціонарний ряд стає стаціонарним. Перші P_s лагів у графіку ЧАКФ перетвореного ряду, відмінні від нуля та повторювані через період s, визначають порядок сезонної складової авторегресії. Перші Q_s лагів у графіку АКФ ряду, відмінні від нуля і повторювані через період s, визначають порядок сезонної складової ковзного середнього.
6. Визначення типу нейромережевої моделі:
   a) У випадку стаціонарності часового ряду
   $$y_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-n}), \quad (1)$$
   де $n = max(p, q)$, а параметри p і q визначені на підставі АКФ та ЧАКФ вихідного часового ряду;
   b) У разі нестаціонарності часового ряду та відсутності сезонності
   $$y_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-n}), \quad (2)$$
   де $n = max(p, q)$, а параметри p і q визначені на підставі АКФ і ЧАКФ часового ряду, перетвореного до стаціонарного ряду методом кінцевих різниць;
   c) У разі нестаціонарності часового ряду та за наявності сезонності

$$y_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-n}, y_{t-s-1}, y_{t-s-2}, \dots, y_{t-s-n}, \dots, y_{t-ms-1}, y_{t-ms-2}, \dots, y_{t-ms-n}, u_t) \quad (3)$$

де $n = max(p, q)$, $m = max(P_S, Q_S)$, s - період сезонності, u_t -індекс сезонності, що обчислюється на основі вихідного часового ряду за формулою

$$u_t = \frac{Y_{st}}{Y_{s0}}$$

тут $Y_{st} = \frac{\sum_{i=1}^{k} Y_{it}}{k}$, середня по кожному внутрішньосезонному періоду t (наприклад, місяцю) для всіх k сезонів (наприклад, років),

$Y_{s0} = \frac{\sum_{i=1}^{k}\sum_{t=1}^{p} Y_{it}}{kp}$ - загальна середня по всіх сезонах (k) та внутрішньосезонних періодів (p), Y_{it} — значення часового ряду для внутрішньосезонного періоду t (наприклад, місяця) та для поточного сезону i (наприклад, року).

Параметри p, q, , P_S, Q_S визначені на підставі АКФ і ЧАКФ тимчасового ряду, перетвореного до стаціонарного методом кінцевих різниць відповідно для поточних значень тимчасового ряду та його сезонних складових.

7. Формування навчальної вибірки на основі значень вихідного часового ряду:
   a) Для моделі виду $y_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-n})$- вибірка формується як
   $$\{(y_{t-n}, \dots, y_{t-2}, y_{t-1}, y_t), \quad t = \overline{n+1, L}\}; \quad (4)$$
   b) Для моделі виду
   $$y_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-n}, y_{t-s-1}, y_{t-s-2}, \dots, y_{t-s-n}, \dots, y_{t-ms-1}, y_{t-ms-2}, \dots, y_{t-ms-n}, u_t)$$
   вибірка формується як
   $$\left\{\begin{pmatrix} y_{t-ms-n}, \dots, y_{t-ms-2}, y_{t-ms-1}, \dots, y_{t-s-n}, \dots, \\ y_{t-s-n}, \dots, y_{t-s-2}, y_{t-s-1}, \dots, y_{t-n}, \dots, y_{t-2}, y_{t-1}, u_t, y_t \end{pmatrix}, \quad t = \overline{ms+n+1, L}\right\} \quad (5)$$

   Тут параметр L дорівнює максимальній кількості даних (рядів), які є у вихідній таблиці.
8. Визначення кількості шарів, кількості нейронів у шарах та навчання нейронної мережі прямого поширення (перцептрон) методом зворотного розповсюдження помилки.

2.3. Характеристики часових рядів

На сьогоднішній день, математичні моделі, які використовують часові ряди для прогнозування розвитку процесів, є найбільш ефективними.

Часовий ряд – це послідовність вимірювань однієї і тієї ж величини, які здійснюються протягом певного періоду часу, зазвичай через регулярні проміжки часу t, наприклад, щотижнево, щомісячно або щорічно. Позначимо цю послідовність z_t.

Задача пронозу: знаючи значення часового ряду у момент часу t, потрібно спрогнозувати значення цього ряду в момент часу $t + l$, де l – це час упередження [18].

Часовий ряд може бути стаціонарним або нестаціонарним, як і інші випадкові процеси.

Часовий ряд є стаціонарним, якщо його статистичні властивості не змінюються з часом (Рисунок 1), тобто:

- Математичне очікування є постійним протягом часу (немає тенденції);
- Дисперсія є сталою

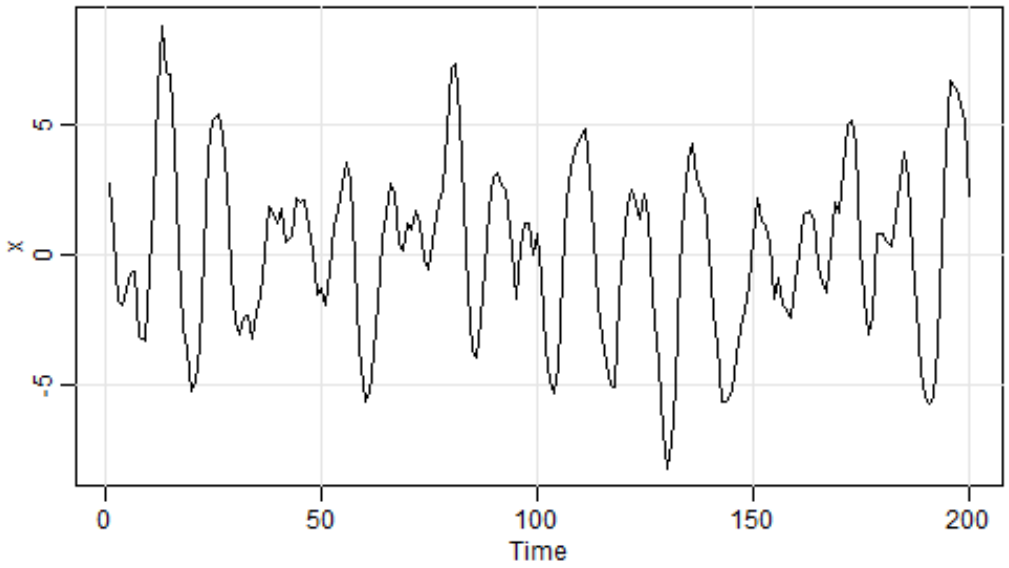

Рисунок 1 Приклад стаціонарного часового ряду

Якщо ж показники математичного сподівання та дисперсії змінюються з часом то ряд буде нестаціонарним (Рисунок 2).

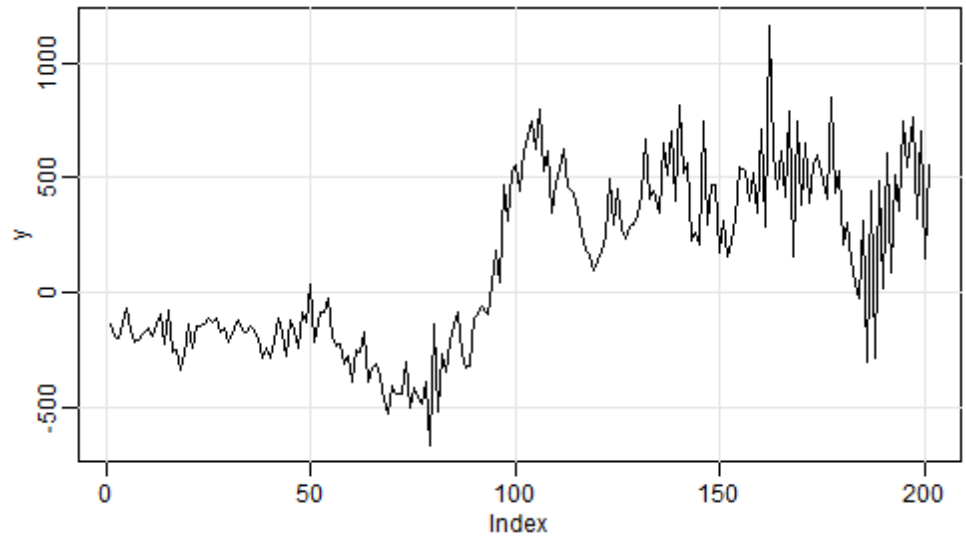

Рисунок 2 Приклад нестаціонарного часового ряду

Аналіз часового ряду починається з виділення його складових компонент, а насамперед тренду [19].

Наявність трендового компонента можна побачити провівши аналіз графіку часового ряду. На графіку це буде виглядати як дуже повільне зростання чи зменшення показників ряду протягом певного періоду часу. Наявність тренду пояснюється демографічними змінами, змінами в структурі виробництва, попиту та інше. Дія подібних факторів є постійною.

Існує також сезонна компонента, яка відображає коливання навколо трендової складової. Наявність такої складової пояснюється сезонним характером споживання чи виробництва. Основна ідея виділення сезонних складових полягає у порівнянні значень часового ряду за певні періоди. Для прикладу, дані за серпень одного року треба порівнювати з даними за серпень минулих років, а не з будь яким іншим місяцем.

Посереднє місце між сезонною складовою та трендом займає циклічний компонент. Оскільки тренд – гладка змінна, що проявляється на досить великому часовому інтервалі, а сезонна компонента – є періодичною функцією, яка залежить від часу (до того ж період сезонної компоненти менший за кількість спостережень), то циклічний компонент представляє собою гладку змінну, яка залежить від часу, але не відноситься ні до сезонної складової, ні до тренду.

Також існує такий компонент як випадковий, він являє собою те, що залишається від часового ряду після того, як були виключені тренд, сезонність та циклічність. В випадку коли в часовому ряді наявний випадковий компонент проноз значень часового ряду буде з похибкою. Ефект випадкового компонента в часовому ряду може бути частково віднесений до стихійних явищ (наприклад,

пожежі), а частково до випадкової діяльності людей. Варто пам'ятати, що будь-який реальний процес має випадковий компонент.

2.4. Класичні моделі прогнозування

Моделі AR, MA, $ARMA$, $ARIMA$ та $SARIMA$ використовуються для прогнозування спостережень $(t+1)$ на основі даних попередніх часових спостережень. Однак необхідно переконатися, що часовий ряд є стаціонарним. Якщо часовий ряд не є стаціонарним, ми можемо застосувати диференціювання часового ряду [20].

Для стаціонарних і нестаціонарних часових рядів існують різні моделі прогнозування.

Для стаціонарних часових рядів:

- Авторегресійна модель $AR(p)$
  Авторегресійна модель $AR(p)$ являє собою модель часових рядів , в якій значення часового ряду в даний момент часу лінійно залежать від попередніх значень цього ж ряду.
  Авторегресійний процес порядку p записується так [21]:

$$Y_t = a_0 + \sum_{i=1}^{p} a_i Y_{t-i} + \varepsilon_t \quad (6)$$

  Де a_i – це параметри моделі (коефіцієнти авторегресії), a_0 – константа (для спрощення можна прийняти її рівною нулю), а ε_t-білий шум.
- Модель ковзного середнього $MA(q)$
  Модель ковзного середнього $MA(q)$ порядка q описує стаціонарні процеси як певну лінійну комбінацію білого шуму і записується наступним чином:

$$Y_t = \sum_{j=0}^{q} b_j \varepsilon_{t-j} \quad (7)$$

  Де ε_t- білий шум, b_j – це параметри моделі (b_0 можно вважати рівним 1) [22].
- Модель авторегресії – ковзного середнього $ARMA(p,q)$
  Модель авторегресії-ковзного середнього $ARMA(p,q)$ забезпечує опис стаціонарного стохастичного процесу в термінах двох поліномів, один для авторегресії, а другий для ковзного середнього [23]. Це є поєднанням моделей $AR(p)$ та $MA(q)$. Модель $ARMA(p,q)$,

де p та q – цілі числа, що задають порядок моделі, записується у такому вигляді:

$$Y_t = c + \varepsilon_t + \sum_{i=1}^{p} a_i Y_{t-i} + \sum_{j=1}^{q} b_j \varepsilon_{t-j} \quad (8)$$

Де c – це константа, ε_t- білий шум, $a_1, \dots, a_p$ – дійсні числа, коефіцієнти авторегресії, $b_1, \dots, b_q$- дійсні числа, коефіцієнти ковзного середнього.

Для нестаціонарних часових рядів:

- Інтегрована модель авторегресії-ковзного середнього $ARIMA(p, d, q)$
  Модель $ARIMA(p, d, q)$ є розширенням моделі $ARMA(p, q)$ для нестаціонарних часових рядів, які можна зробити стаціонарними взявши різність певного порядку від вихідного часового ряду. Модель $ARIMA(p, d, q)$ описується наступною формулою:

$$\Delta^d Y_t = c + \sum_{i=1}^{p} a_i \Delta^d Y_{t-i} + \sum_{j=1}^{q} b_j \varepsilon_{t-j} + \varepsilon_t \quad (9)$$

  Де c – це константа, ε_t- білий шум, $a_1, \dots, a_p$ – дійсні числа, коефіцієнти авторегресії, $b_1, \dots, b_q$- дійсні числа, коефіцієнти ковзного середнього, Δ^d- оператор різності часового ряду порядку d.

- Сезонна інтегрована модель авторегресії- ковзного середнього $SARIMA(p, d, q)(P_s, D_s, Q_s)$
  Моделі $SARIMA(p, d, q)(P_s, D_s, Q_s)$ корисні для моделювання сезонних часових рядів, у яких середнє значення та інші статистичні дані для певного сезону не є стаціонарними протягом багатьох років. Визначена модель $SARIMA(p, d, q)(P_s, D_s, Q_s)$ є прямим розширенням представлених моделей авторегресії - ковзного середнього $ARMA(p, q)$ та інтегрованих моделей авторегресії - ковзного середнього $ARIMA(p, d, q)$.

$$\Delta_s^D \Delta^d Y_t = c + \sum_{i=1}^{p} a_i \Delta_s^D \Delta^d Y_{t-i} + \sum_{j=1}^{q} b_j \varepsilon_{t-j} + \varepsilon_t \quad (10)$$

  Де c – це константа, ε_t- білий шум, $a_1, \dots, a_p$ – дійсні числа, коефіцієнти авторегресії, $b_1, \dots, b_q$- дійсні числа, коефіцієнти ковзного середнього, Δ^d- оператор різності часового ряду порядку d, Δ_s^D- оператор різності часового ряду за сезонністю порядку D.

2.5. Автокореляційна функція та часткова автокореляційна функція

Автокореляція та часткова автокореляція слугують мірою зв'язаності між поточними значеннями ряду та минулими і показують, які минулі значення є найбільш корисними для прогнозування майбутніх значень.

Автокореляційною функцією (АКФ) є функція оцінки коефіцієнта автокореляції в залежності від величини часового лагу між досліджуваними рядами.

$$r_\tau = r_{yt\ yt+\tau} = \frac{\sum_{t=1}^{T-\tau}(y_t - \bar{y})(y_{t+\tau} - \bar{y})}{\sum_{t=1}^{T}(y_y - \bar{y})^2},$$
$$\bar{y} = \frac{1}{T}\sum_{t=1}^{T} y_t \qquad (11)$$

Графічно функція автокореляції зображується за допомогою корелограми. Корелограма відображає графічно й чисельно коефіцієнти автокореляції та їх стандартні похибки для послідовності лагів з певного діапазону [24]. Часовий ряд може мати такі компоненти як тренд, сезонність, циклічність та шум. Автокореляційна функція враховує всі ці компоненти при пошукі кореляцій [25], саме тому їх можна побачити на графіку (Рисунок 3).

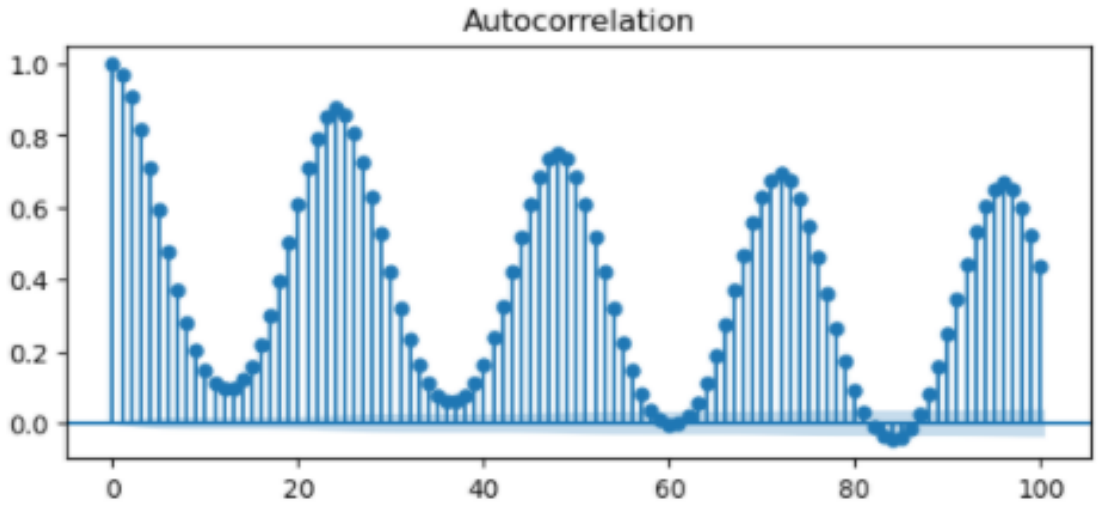

Рисунок 3 Корелограма автокореляційної функції

Часткова автокореляційна функція (ЧАКФ) – це більш поглиблена версія звичайної автокореляційної функції. Відмінність полягає лише в виключенні

кореляційної залежності між спостереженнями всередині лагів. Тобто часткова автокореляційна функція на кожному лагі відрізняється від звичайної на величну видалених автокореляцій з меншими часовими лагами. Із цього слідує, що часткова автокореляційна функція більш точно характеризує автокореляційні залежності всередині часового ряду (Рисунок 4).

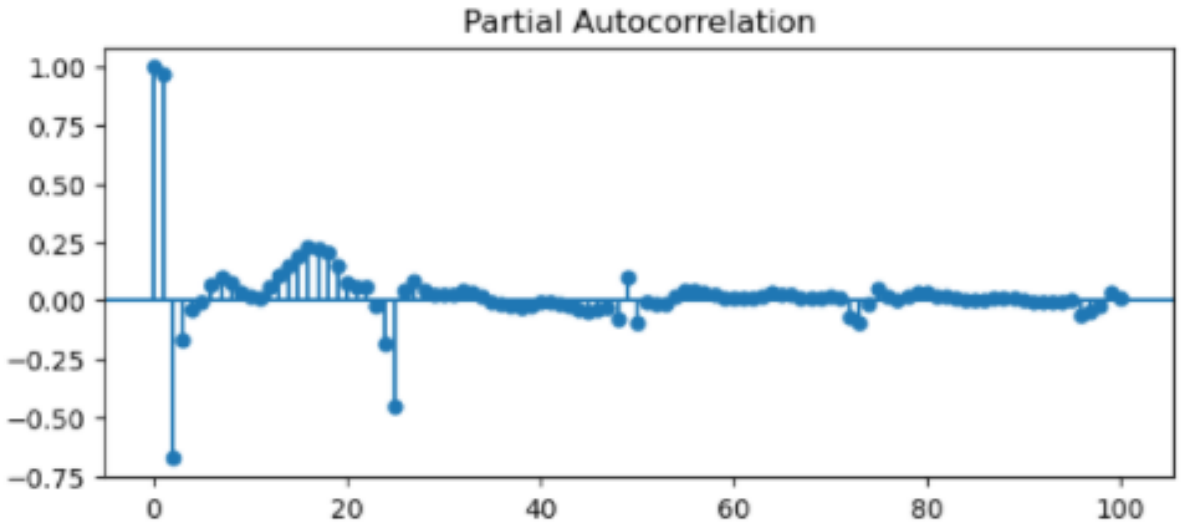

Рисунок 4 Корелограма часткової автокореляційної функції

3. НЕЙРОННА МЕРЕЖА

3.1. Нейронна мережа прямого поширення

Нейронна мережа прямого поширення – це штучна нейронна мережа (Рисунок 5). В такій мережі зв'язки між вузлами не утворюють цикл. Нейронна мережа прямого поширення була першим і найпростішим типом нейронних мереж.

В мережі подібного типу інформація переміщується по напрямку вперед від вхідних вузлів та через приховані вузли (якщо такі звичайно є) до вихідних вузлів.

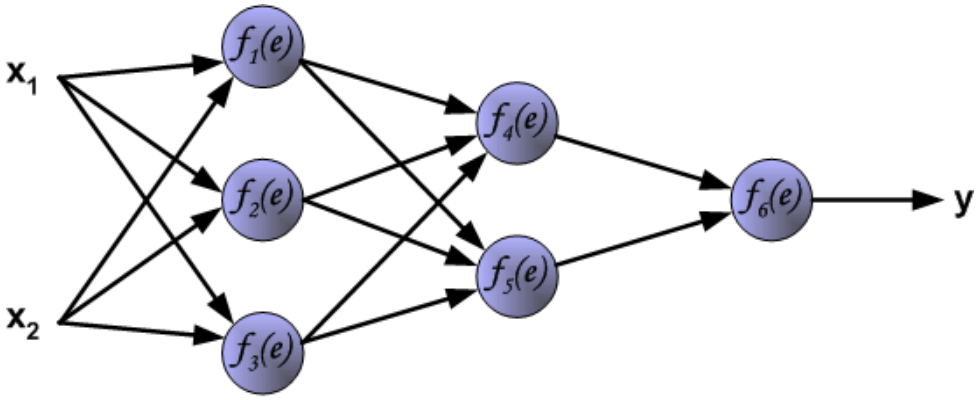

Рисунок 5 Приклад нейронної мережі з двома вхідними нейронами [29]

Розглянемо нейронну мережу прямого поширення на прикладі [27], де є два вхідних сигнали x_1 і x_2 та вихідний сигнал y:

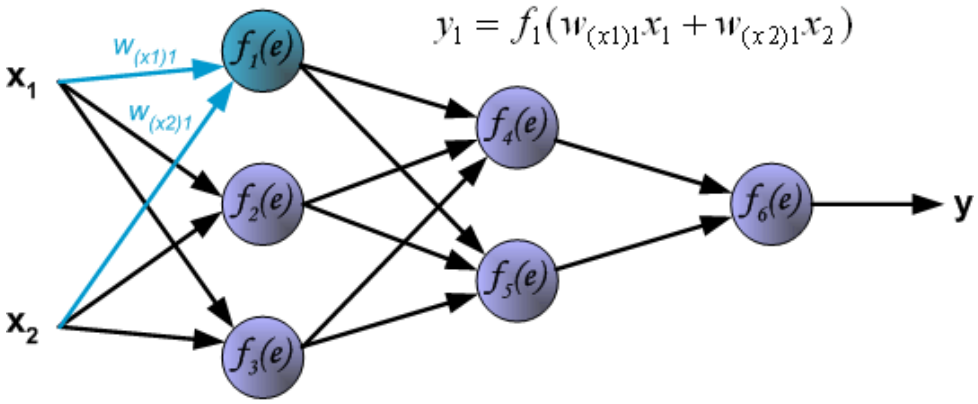

Рисунок 6 Приклад поширення сигналу з вихідного шару на прихований [29]

Таким же самим чином, як наведено в рисунку 6 розраховується значення сигналу для $y_2 = f_2(e)$ та $y_3 = f_3(e)$.

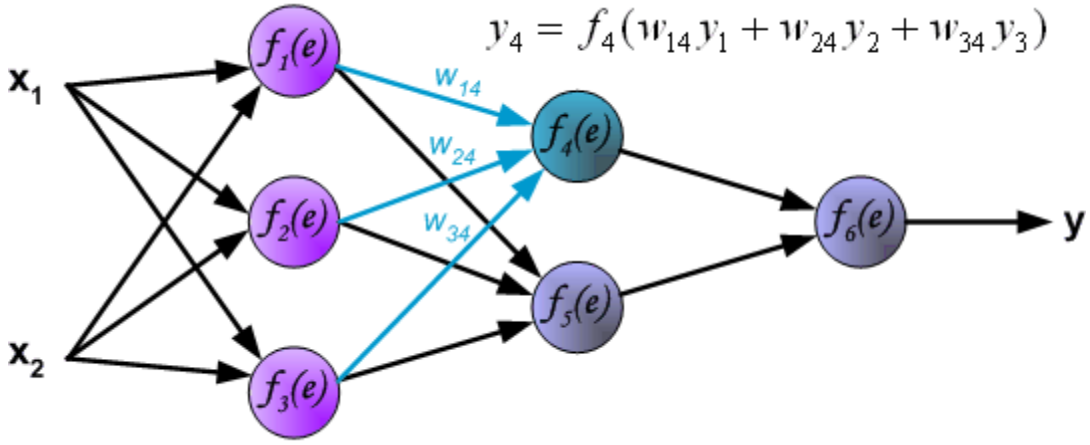

Рисунок 7 Приклад поширення сигналу через прихований шар [29]

Значення для $y_5 = f_5(e)$ розраховується так само як наведено в рисунку 7, змінюються тільки відповідні значення вагів.

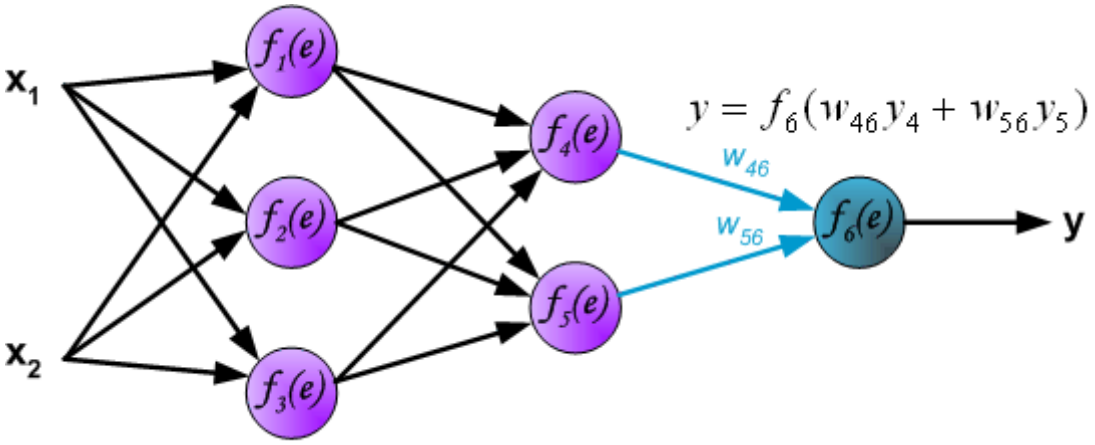

Рисунок 8 Приклад поширення сигналу через вихідний шар [29]

3.2. Навчання нейронної мережі прямого поширення методом зворотного розповсюдження помилки

Ймовірно, метод зворотного розповсюдження помилки є самою фундаментальною складовою нейронної мережі.

Метод зворотного розповсюдження помилки використовується для ефективного навчання нейронної мережі за допомогою ланцюгового правила (правила диференціювання складної функції). Говорячи іншими словами, після кожного проходження по мережі зворотне розповсюдження рухається в зворотному напрямку і регулює такі параметри моделі як ваги та зміщення.

На прикладі простої мережі з двома нейронами на вході і одним на виході розглянемо найскладніший, з точки зору математики, момент алгоритму навчання нейронної мережі – розрахунок градієнта функції вартості.

Просумувавши добутки входів та вагових коефіцієнтів та пропустивши отримані суми через активаційну функцію на виході нейронної мережі результатом отримуємо y.

Спершу отриманий результат y буде відрізнятися від бажаного z. Похибка визначається за формулою (7):

$$\delta = z - y \tag{12}$$

На основі похибки δ розраховується градієнт функції вартості для вхідного вектора $x = (x_1, x_2)$:

$$C = \frac{1}{2}(y - z)^2 \tag{13}$$

Згідно правила оновлення градієнтного спуску (9)

$$\omega_k' = \omega_k - \eta \frac{\partial C}{\partial \omega_k}, \qquad b_l' = b_l - \eta \frac{\partial C}{\partial b_l} \tag{14}$$

всі ваги і зміщення корегуються однаково. Однак, корегування вагових коефіцієнтів зовнішнього шару та прихованих шарів дещо відрізняються.

Корегування вагових коефіцієнтів для зовнішнього шару:

Розрахуємо $\frac{\partial C}{\partial \omega_{46}}$, використовуючи ланцюгове правило.

$$\frac{\partial C}{\partial \omega_{46}} = \frac{\partial C}{\partial y} \cdot \frac{\partial y}{\partial e_6} \cdot \frac{\partial e_6}{\partial \omega_{46}}, \quad (15)$$

Де $e_6 = w_{46}y_4 + w_{56}y_5$ – сумарний сигнал на вході 6-го нейрона.

$$\frac{\partial C}{\partial y} = (y - z) = \delta \quad (16)$$

з формули (13).

$$\frac{\partial y}{\partial e_6} = f_6', \quad (17)$$

оскільки $y = f\_6(e)$.

$$\frac{\partial e_6}{\partial \omega_{46}} = y_4. \quad (18)$$

Із цього слідує що,

$$\frac{\partial C}{\partial \omega_{46}} = \delta f_6' y_4. \quad (19)$$

Загальний вигляд правила корегування вагових коефіцієнтів зовнішнього шару:

$$\omega_{ji} = \omega_{ji} - \eta \delta_i f_i' y_i, \quad (20)$$

де j- номер нейрона прихованого шару, i-номер нейрона вихідного шару, δ_i – похибка на i-му нейроні.

Формула для корегування вагових коефіцієнтів нейронів внутрішнього шару k мережі виглядає наступним чином:

$$\omega_{ji}^{(k)} = \omega_{ji}^{(k)} - \eta \delta_i^{(k)} y_j^{(k)} = \omega_{ji}^{(k)} - \eta \left[\sum_p \delta_p^{(k+1)} \omega_{ip}^{(k+1)} \right] \left(f_i^{(k)} \right)' y_j^k, \quad (21)$$

де j – номер нейрона k-го шару, i – номер нейрона $k+1$ шару.

4. РЕЗУЛЬТАТИ

4.1. Аналіз часового ряду енергоспоживання

Для роботи, з відкритого ресурсу Kaggle були взяті дані про погодинне споживання електроенергії в Сполучених Штатах [30]. Нижче наведено зразок даних:

	Datetime	DUQ_MW
0	2005-12-31 01:00:00	1458.0
1	2005-12-31 02:00:00	1377.0
2	2005-12-31 03:00:00	1351.0
3	2005-12-31 04:00:00	1336.0
4	2005-12-31 05:00:00	1356.0

Рисунок 9 Зразок даних споживання електроенергії

Головною метою розкладання часових рядів є розкладання рівнів ряду на складові компоненти з метою врахування їх при прогнозуванні, тобто декомпозиція часового ряду.

Оскільки значень в ряді дуже багато, для візуалізації ряду були взяті дані за місяць.

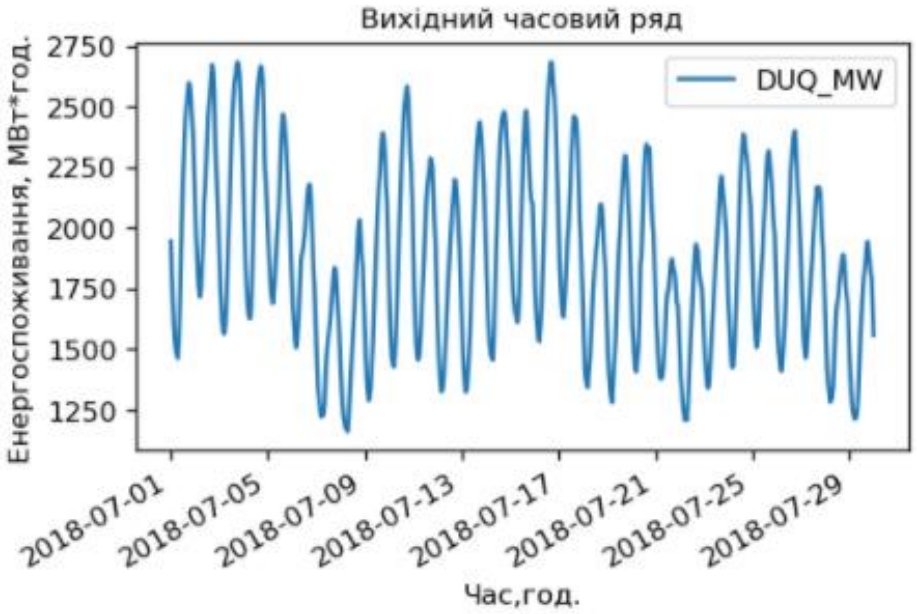

Рисунок 10 Графік значень часового ряду

З наведеного вище рисунку видно, що в ряді наявні тренд та сезонність.

Для більшої наочності виділяємо кожну складову окремо. Аналізуючи часовий ряд перш за все виявляють тенденцію (тренд) , яка визначає основний напрямок розвитку явища за великий проміжок часу.

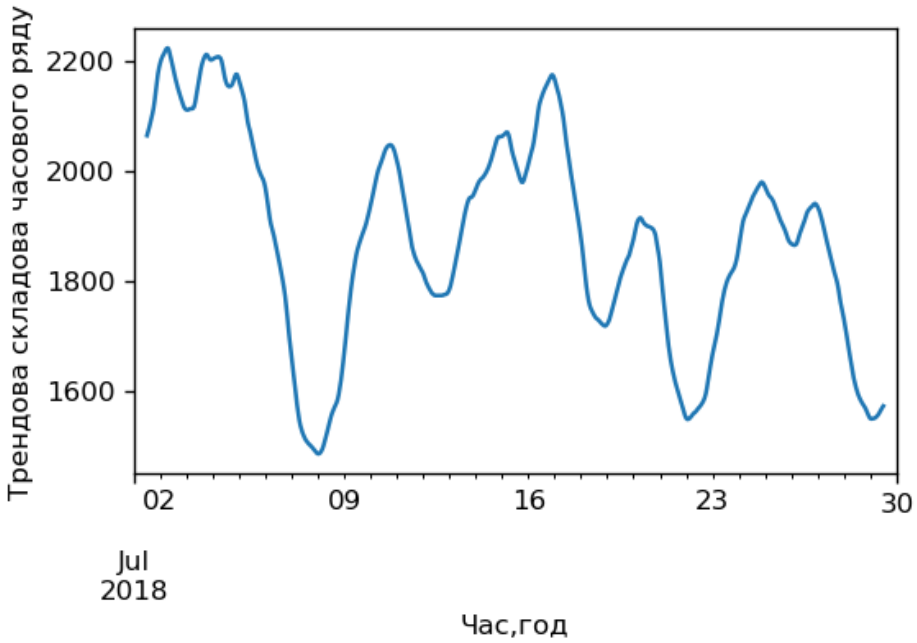

Рисунок 11 Трендова складова часового ряду

Далі виділяємо сезонну складову:

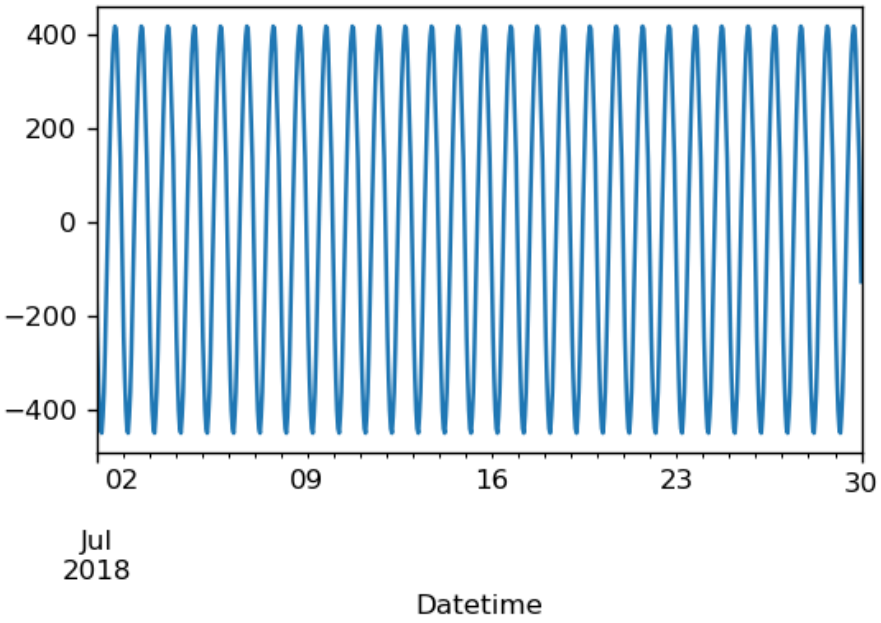

Рисунок 12 Сезонна складова часового ряду

І останнім виділяємо шум:

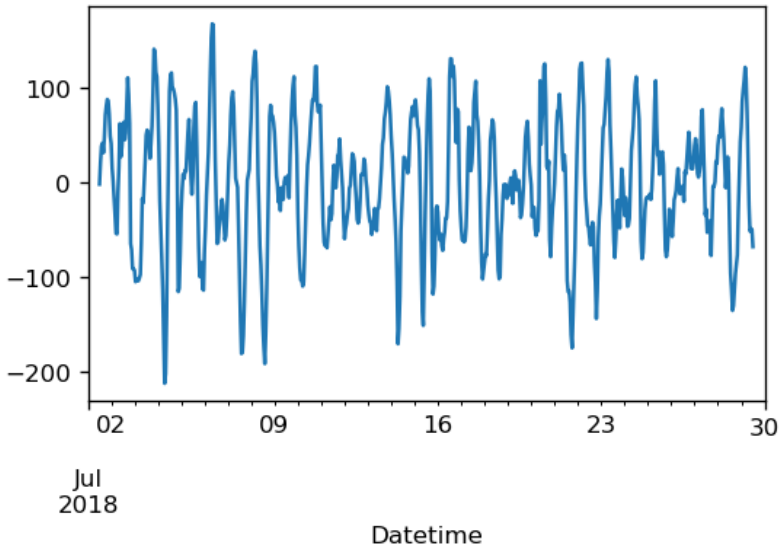

Рисунок 13 Шумова складова часового ряду

Виходячи з отриманих результатів вже можна сказати, що ряд є нестаціонарним і присутня сезонність. Але більшої впевненості проводимо тест Дікі-Фуллера [28]. Результатом такого тесту є значення $p - value$, дивлячись на яке можна остаточно зробити висновок про стаціонарність ряду. У випадку, коли $p - value < 0.05$ – ряд є стаціонарним, в іншому випадку – нестаціонарним.

```
ADF Statistic: -1.822604
p-value: 0.369299
```

Рисунок 14 Результати тесту Дікі-Фуллера

Результати тесту підтверджують припущення про нестаціонарність ряду.

Оскільки ряд не є стаціонарним та має сезонність необхідно для прогнозування застосовувати модель $SARIMA(p, d, q)(P_s, D_s, Q_s)$

Для визначення розміру сезонності побудуємо графік автокореляції.

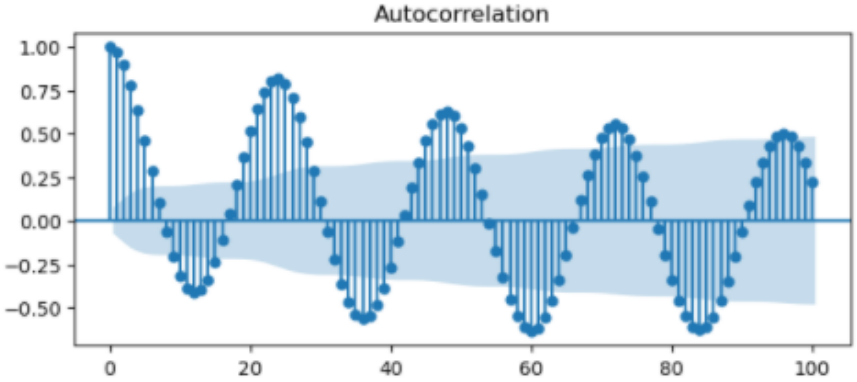

Рисунок 15 Графік автокореляції для нестаціонарного часового ряду

З графіка бачимо, що сезонність складає 24 години, тобто добу.

Оскільки було з’ясовано, що ряд не є стаціонарним, то треба привести його до стаціонарного виду методом взяття кінцевих різниць, як для поточних значень ряду, так і для його сезонних складових.

Маємо наступні результати:

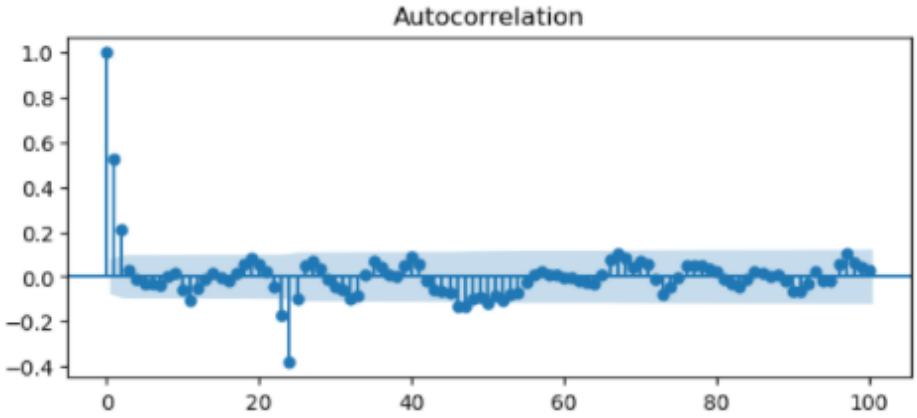

Рисунок 16 Графік автокореляції для стаціонарного ряду

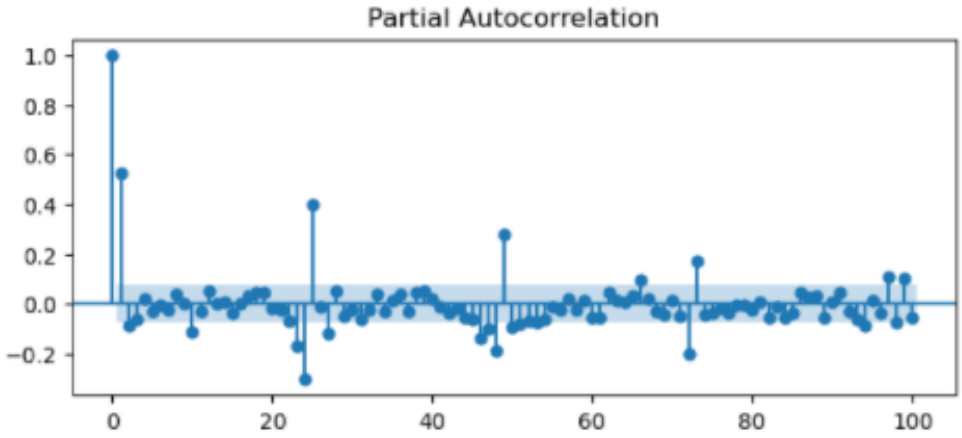

Рисунок 17 Графік часткової автокореляції для стаціонарного ряду

З рис. 16, 17 можемо бачити, що вдалося позбутися сезонності.

З отриманих графіків можна визначити параметри моделі прогнозування $SARIMA(p,d,q)(P_s,D_s,Q_s)$.

З графіку автокореляції можна визначити параметри q та Q_s.

Значення $q = 3$, тому що число 3 відповідає значенню лагу , при якому вперше перетинається верхній довірчий інтервал. Значення $Q_s = 2$, тому що, кількість вагомих лагів, що кратні тривалості сезону дорівнює 2.

Значення $p = 1$, $P_s = 4$ визначаються за аналогічною логікою, але з графіка часткової автокореляції.

Значення $d = 1, D_s = 1$, оскільки після взяття різниць відповідних порядків, часовий ряд став стаціонарним.

4.2. Результати роботи нейронної мережі

На основі отриманих параметрів моделі прогнозування $SARIMA(p,d,q)(P_s,D_s,Q_s)$ формуємо навчальну вибірку (Таблиця *1*) для всіх вхідних даних, починаючи з $t = ms + n + 1$, відповідно до формули (5).

Таблиця 1 Зразок навчальної вибірки

1	y_{t-1}	$y_{99-1=98}$	1395
2	y_{t-2}	$y_{99-2=97}$	1428
3	y_{t-3}	$y_{99-3=96}$	1487
4	y_{t-s-1}	$y_{99-24-1=74}$	1395
5	y_{t-s-2}	$y_{99-24-2=73}$	1428
6	y_{t-s-3}	$y_{99-24-3=72}$	1487
7	y_{t-2s-1}	$y_{99-2*24-1=50}$	1366
8	y_{t-2s-2}	$y_{99-2*24-2=49}$	1399
9	y_{t-2s-3}	$y_{99-2*24-3=48}$	1467
10	y_{t-ms-1}	$y_{99-4*24-1=2}$	1351
11	y_{t-ms-2}	$y_{99-4*24-2=1}$	1377
12	y_{t-ms-3}	$y_{99-4*24-3=0}$	1458
13	u_t	u_{99}	1.07015
14	y_t	y_{99}	1379

Кожна з сформованих начальних вибірок являє собою вхідний шар нейронної мережі (за виключенням значення y_t, яке є цільовою змінною) .

Нейронна мережа прямого поширення була навчена методом зворотного розповсюдження помилки з використанням функції активації LeakyReLU. Нейронна мережа складається з 4-ч шарів (1 вхідний, 2 прихованих, 1 вихідний), які мають по 13, 200, 250 та 1 нейрон відповідно.

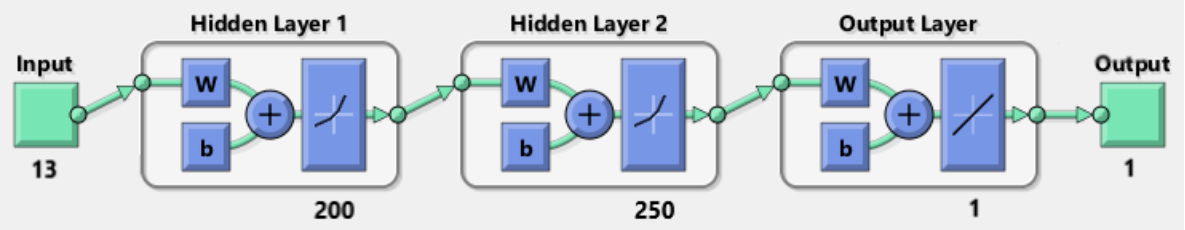

Рисунок 18 Схема 4-х шарової нейронної мережі

В результаті роботи нейронної мережі маємо наступні результати:

Точність для тестового набору (Рисунок 19) даних складає 93.0492%.

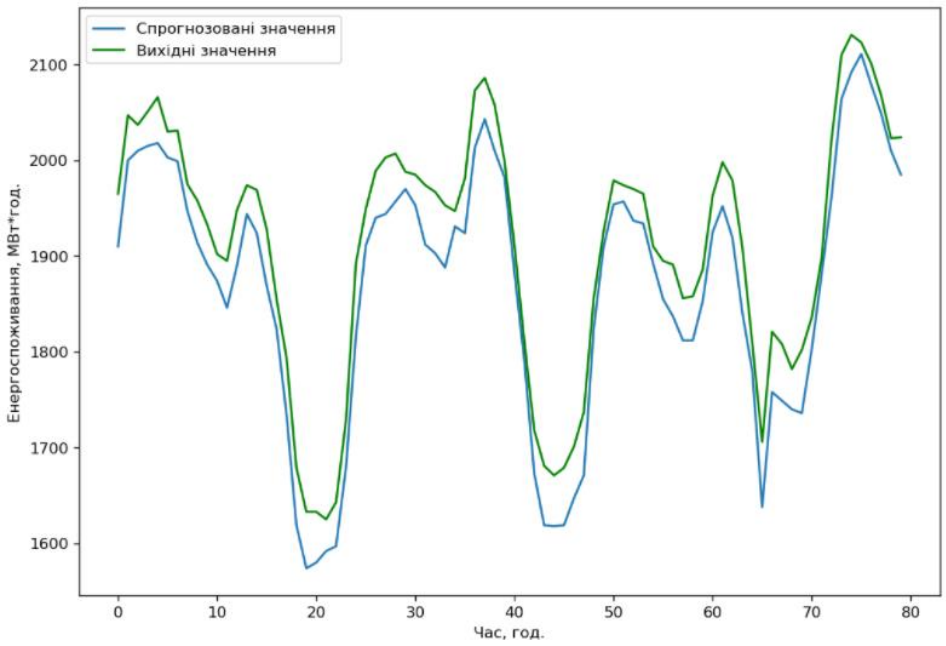

Рисунок 19 Прогноз для тестового набору

Із графіку видно, що наявне систематичне відхилення. Тому знаходимо $\Delta y = y_{real} - y_{forecast} = 46.024$. Додавши середнє відхилення до прогнозованих значень маємо наступні результати:

В результаті подібних маніпуляцій точність підвищилася до 97%.

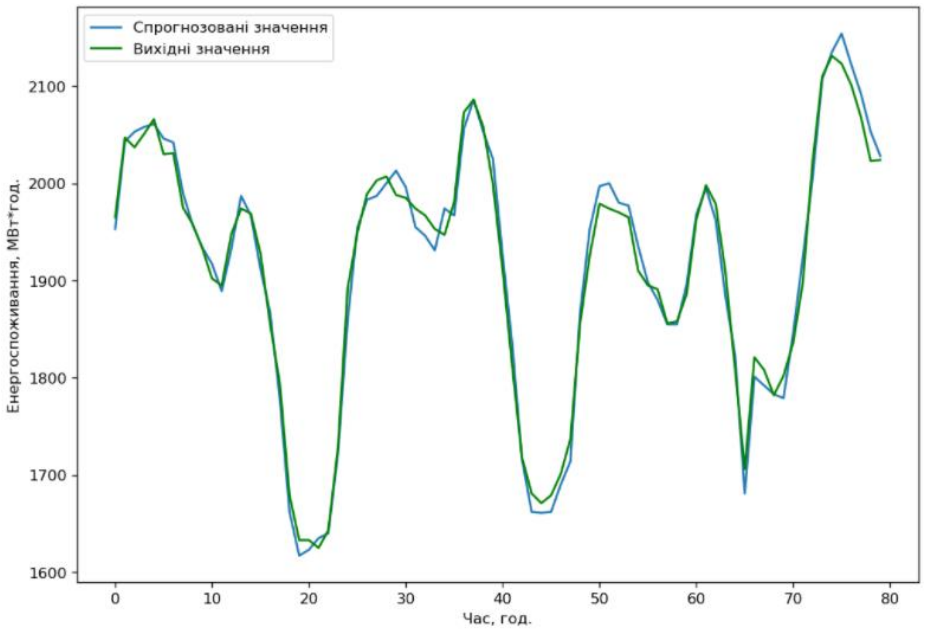

Рисунок 20 Корегований прогноз для тестового набору

4.3. Порівняння результатів нейронної мережі та моделі прогнозування SARIMA

В ході виконання роботи було проведено порівняння результатів роботи нейронної мережі та моделі прогнозування SARIMA для часового ряду без пропусків та з пропусками.

Для даних без пропусків результати роботи нейронної мережі наведено в пункті 4.2.

Результати моделі SARIMA для набору даних без пропусків, наступні:

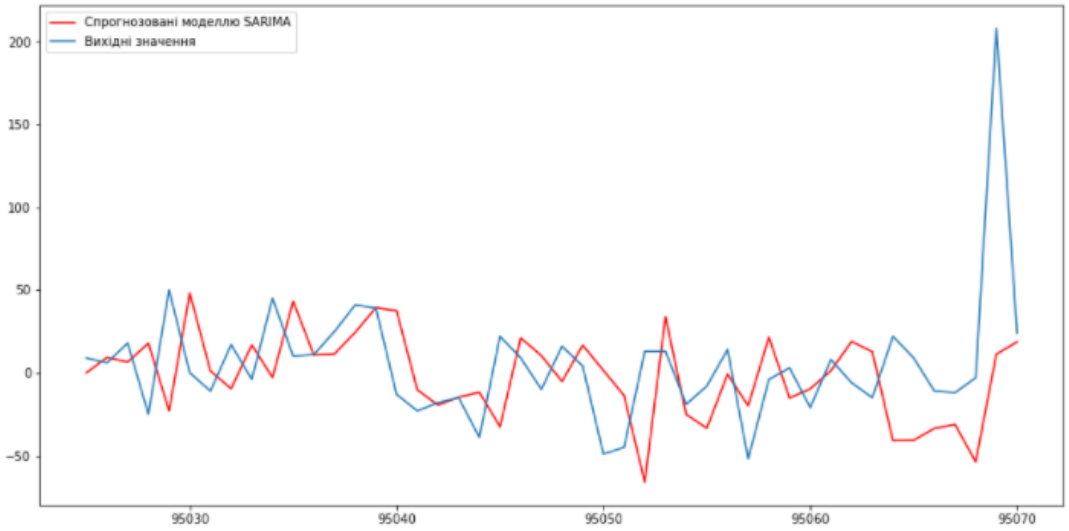

Рисунок 21 Результат роботи моделі SARIMA для даних без пропусків

Точність прогнозу складає 96.62%.

Результати роботи нейронної мережі для набору даних з пропусками виглядають наступним чином:

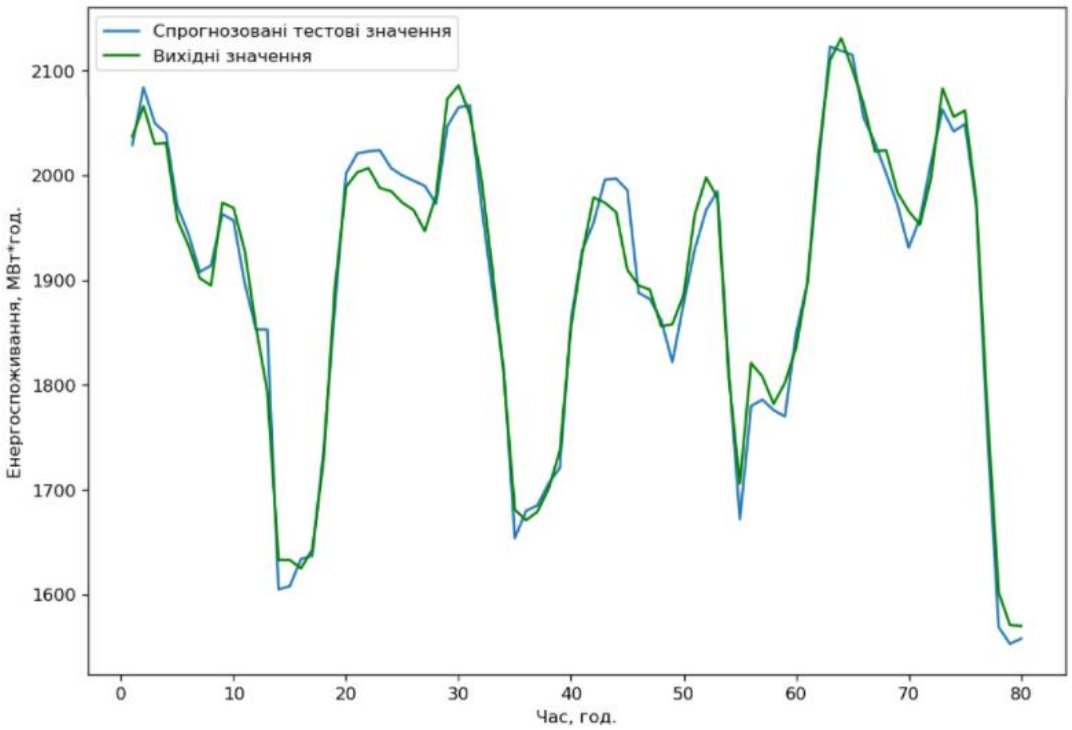

Рисунок 22 Прогноз нейронної мережі для даних з пропусками

Точність такого прогнозу становить 96%.

Результати прогнозної моделі SARIMA для даних з пропусками мають вигляд, як показано на рис.23:

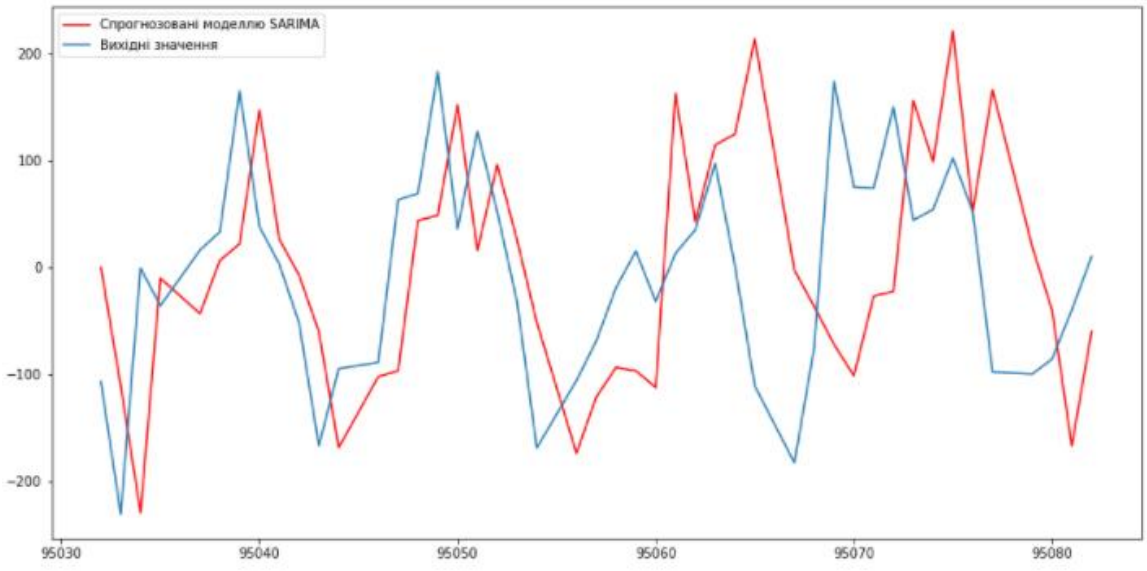

Рисунок 23 Результат роботи моделі SARIMA для даних з пропусками

Точність становить 91%.

ВИСНОВКИ

В роботі було розглянуто проблему прогнозування часових рядів. Було побудовано нейронну мережу прямого поширення, яка була навчена за допомогою методу зворотного розповсюдження помилки. Були розглянуті нейромережеві моделі прогнозування стаціонарних та нестаціонарних часових рядів.

Поставлене завдання було виконане у такій послідовності:

- Проведено аналіз часового ряду;
- Визначено вид моделі прогнозування;
- Знайдено параметри моделі прогнозування SARIMA;
- Визначено вид нейромережевої моделі, а саме нейромережева модель для нестаціонарного часового ряду з наявною сезонністю;
- Сформовано навчальну вибірку відповідно до виду нейромережевої моделі, на основі значень вихідного часового ряду;
- Побудовано нейронну мережу прямого поширення методом зворотного розповсюдження помилки;
- Порівняно результати роботи нейронної мережі та моделі прогнозування SARIMA.

В результаті виконання роботи можна зробити такі висновки:

- Для виконання роботи було застосовано модель типу $SARIMA$, оскільки в часовому ряді була наявна сезонна компонента.
- Нейронна мережа має кращі результати прогнозування в порівнянні з моделлю прогнозування SARIMA;
- На відміну від моделі прогнозування SARIMA, нейронна мережа має більшу точність прогнозу для набору даних з пропусками;

Основною метою роботи було написання нейронної мережі, яка б мала можливість робити прогноз для часового ряду з пропусками. Мета була досягнута.

СПИСОК ЛІТЕРАТУРИ

1. Ухатов А. Р. Разработка модели прогнозирования временных рядов нестационарных дискретных систем на основе нейронной сети / А. Р. Ухатов, Ф. М. Назаров. // журнал Проблемы информатики. – 2018. – №3. – С. 34–50.
2. Зуева В. Н. Нейросетевой модуль прогнозирования потребления электроэнергии / В. Н. Зуева, Д. А. Трухан, Д. Н. Карлов. // Политематический сетевой электронный научный журнал Кубанского государственного аграрного университета. – 2017. – №132. – С. 1322–1331.
3. Ткалич С. А. Концепция безаварийного управления на основе моделей прогнозирования состояний потенциально опасных технологических процессов / С. А. Ткалич, В. Л. Бурковский, О. Ю. Таратынов. // Вестник Воронежского государственного технического университета. – 2016. – №6. – С. 79–86.
4. Селиверстова А. В. Сравнительный анализ моделей и методов прогнозирования / А. В. Селиверстова. // Современные научные исследования и инновации. – 2016. – №11. – С. 241–248.
5. Jiang P. A hybrid forecasting system based on fuzzy time series and multi-objective optimization for wind speed forecasting / P. Jiang, H. Yang, J. Heng. // Applied Energy. – 2019. – №235. – С. 786–801.
6. Nguyen L. Forecasting seasonal time series based on fuzzy techniques / L. Nguyen, V. Novak. // Fuzzy Sets and Systems. – 2019. – №361. – С. 114–129.
7. Емалетдинова Л. Ю. Методика нейросетевого прогнозирования на основе анализа временных рядов / Л. Ю. Емалетдинова, З. И. Мухаметзянов. // Передовые инновационные разработки. Перспективы и опыт использования, проблемы внедрения в производство: сборник научных статей III Международной научной конференции. – 2019. – С. 16–17.
8. Jinu L. A neural network method for nonlinear time series analysis / L. Jinu. // Journal of Time Series Econometrics. – 2019. – №11. – С. 1–18.
9. Singhal D. Electricity price forecasting using artificial neural networks / D. Singhal, K. S. Swarup. // International Journal of Electrical Power and Energy Systems. – 2011. – №3. – С. 550–555.
10. Нейросетевые модели с виртуальными потоками для классификации и пронозирования функционального состояния сложных систем / [А. В. Киселев, Т. В. Петрова, С. В. Дегтярев та ін.]. // Известия Юго-Западного государственного университета. – 2018. – №4. – С. 123–134.
11. Сравнительный анализ итерационного и прямого нейросетевого краткосрочного прогнозирования электропотребления крупного города / И.

Е.Шепелев, И. И. Надтока, С. А. Вялкова, С. О. Губский. // Нейрокомпьютеры: разработка, применение. – 2016. – №3. – С. 21–30.
12. Шолтанюк С. В. Сравнительный анализ нейросетевой и регрессионных моделей прогнозирования временных рядов / С. В. Шолтанюк. // Цифровая трансформация. – 2019. – №2. – С. 60–68.
13. Bandara K. Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach / K. Bandara, C. Bergmeir, S. Smyl. // Expert Systems with Application. – 2020. – №140.
14. Kumar D. N. River flow forecasting using recurrent neural networks / D. N. Kumar, K. S. Raju, T. Sathish. // Water Resources Management. – 2004. – №2. – С. 143–161.
15. Захаров А. А. Интеллектуальный модуль анализа данных в информационных системах с помощью искусственных нейронных сетей / А. А. Захаров, Е. А. Оленников, Т. И. Паюсова. // Вестник Тюменского государственного университета. Физико-математическое моделирование.. – 2015. – №4. – С. 102–111.
16. Катасёв А. С. Нейронечеткая модель формирования нечетких правил для оценки состояния объектов в условиях неопределенности / А. С. Катасёв. // Компьютерные исследования и моделирование. – 2019. – №3. – С. 477–492.
17. Трегуб А. В. Методика построения модели ARIMA для прогнозирования динамики временных рядов / А. В. Трегуб, И. В. Трегуб. // Лесной вестник. – 2011. – №5. – С. 179–183.
18. Математичні методи ідентифікації динамічних систем [Електронний ресурс] // Розділ 5 Стохастичні моделі дискретних лінійних динамічних систем з зосередженими параметрами на основі часових рядів – Режим доступу до ресурсу: https://web.posibnyky.vntu.edu.ua/feeem/1mokin_matmetody_identifikaciyi_dinamsystem/5-1.html.
19. Ставицький А. В. Теорія часових рядів [Електронний ресурс] / А. В. Ставицький – Режим доступу до ресурсу: http://andriystav.cc.ua/Downloads/AppliedEco/02_Time_Series.pdf.
20. Shetty C. Time Series Models [Електронний ресурс] / C. Shetty // Towars Data Scienes. – 2020. – Режим доступу до ресурсу: https://towardsdatascience.com/time-series-models-d9266f8ac7b0.
21. Авторегрессионная модель [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%BE

%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C.
22. Модель скользящего среднего [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D1%81%D0%BA%D0%BE%D0%BB%D1%8C%D0%B7%D1%8F%D1%89%D0%B5%D0%B3%D0%BE_%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B5%D0%B3%D0%BE.
23. Модель авторегрессии — скользящего среднего [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B0%D0%B2%D1%82%D0%BE%D1%80%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D0%B8_%E2%80%94_%D1%81%D0%BA%D0%BE%D0%BB%D1%8C%D0%B7%D1%8F%D1%89%D0%B5%D0%B3%D0%BE_%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B5%D0%B3%D0%BE.
24. Автокорреляционная функция (АКФ) и ее свойства. Частная автокорреляционная функция (ЧАКФ) и ее свойства [Електронний ресурс] // studopedia. – 2015. – Режим доступу до ресурсу: https://studopedia.ru/8_30957_avtokorrelyatsionnaya-funktsiya-akf-i-ee-svoystva-chastnaya-avtokorrelyatsionnaya-funktsiya-chakf-i-ee-svoystva.html.
25. Значение графиков ACF и PACF в анализе временных рядов [Електронний ресурс] // machinelearningmastery. – 2019. – Режим доступу до ресурсу: https://www.machinelearningmastery.ru/significance-of-acf-and-pacf-plots-in-time-series-analysis-2fa11a5d10a8/.
26. Нейронная сеть с прямой связью [Електронний ресурс] // Wikipedia. – 2021. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C_%D1%81_%D0%BF%D1%80%D1%8F%D0%BC%D0%BE%D0%B9_%D1%81%D0%B2%D1%8F%D0%B7%D1%8C%D1%8E.
27. Нейронная сеть - обучение ИНС с помощью алгоритма обратного распространения [Електронний ресурс] // robocraft – Режим доступу до ресурсу: https://robocraft.ru/blog/algorithm/560.html.
28. Wikipedia. Тест Дікі-Фуллера [Електронний ресурс] / Wikipedia // Wikipedia. – 2021. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%.

29. Mariusz B. Principles of training multi-layer neural network using backpropagation [Електронний ресурс] / B. Mariusz, W. Przemysław. – 2004. – Режим доступу до ресурсу: http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html.
30. Hourly Energy Consumption [Електронний ресурс] – Режим доступу до ресурсу: https://www.kaggle.com/robikscube/hourly-energy-consumption.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
plt.rcParams.update({'figure.figsize': (10, 7),
'figure.dpi': 120})

# Import as Dataframe
df1 = pd.read_csv('DUQ_hourly1.csv')
df1.head()

df1.isnull().sum()

df1.dtypes

df1['Datetime'] = pd.to_datetime(df1['Datetime'])
df1.head()

df1.dtypes

df1 = df1.set_index('Datetime')
df1.head()

from datetime import datetime
start_date = datetime(2018, 7, 1)
end_date = datetime(2018, 7, 30)
df1[(df1.index >= start_date) & (df1.index <=
end_date)].plot(ylabel='Енергоспоживання, МВт*год.',
xlabel='Час,год.', figsize=(5,3))
plt.title("Вихідний часовий ряд", fontsize=10)

df2 = df1[(df1.index >= start_date) & (df1.index <=
end_date)]
dataMonth = df2.sort_values(by='Datetime')

import statsmodels.api as sm
decomposition = sm.tsa.seasonal_decompose(dataMonth,
model='additive')

trend = decomposition.trend
trend.plot(ylabel='Трендова складова часового ряду',
xlabel='Час,год',figsize=(5,3))

seasonal = decomposition.seasonal
```

```
seasonal.plot(figsize=(5,3))

resid = decomposition.resid
resid.plot(figsize=(5,3))

from statsmodels.tsa.stattools import adfuller
result = adfuller(dataMonth)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])

from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf,
plot_pacf
# Draw Plot
fig1, axes1 = plt.subplots(1,2,figsize=(16,3), dpi= 100)
plot_acf(dataMonth, lags=100, ax=axes1[0])
plot_pacf(dataMonth, lags=100, ax=axes1[1])

# Draw Plot
fig1, axes6 = plt.subplots(1,2,figsize=(16,3), dpi= 100)
plot_acf(dataMonth.diff().diff(24).dropna(), lags=100,
ax=axes6[0])
plot_pacf(dataMonth.diff().diff(24).dropna(), lags=100,
ax=axes6[1])

# Neural Network
newDataTrain = df1[0:95000]
newDataTest = df1[95000:119000]

newDataTrain = newDataTrain[['DUQ_MW']]
newDataTrain = newDataTrain.DUQ_MW.tolist()

newDataTest = newDataTest[['DUQ_MW']]
newDataTest = newDataTest.DUQ_MW.tolist()

n = 3
m = 4
s = 24


def StatNonStatSeries(train_mas, n):
    x_large_mas = []
    y_values = []

    indicator = n
```

```
    for i in range(indicator, len(train_mas)):
        y_v = []
        y_v.append(int(train_mas[i]))
        y_values.append(y_v)
        # y_values.append(int(train_mas[i]))
        # print('First cycle iteration i = ', i)
        x_mini_mas = []
        for j in range(1, m + 1):
            # print('Second cycle iteration i = ', i)
            # print('Second cycle iteration j = ', j)
            x_mini_value = train_mas[i - j]
            x_mini_mas.append(int(x_mini_value))
            # print(x_mini_mas)

        x_large_mas.append(x_mini_mas)

    return x_large_mas, y_values

def YearsCounter(train_mas):
    count = 0
    for i in range(len(train_mas)):
        if i% (365 * 24) == 0:
            count = count + 1
            #print(i)
    return count


def NonStatSeries(train_mas, n, m, s):
    ns_b = []
    ss_b = []
    ms_b = []

    y_values = []

    mas_large = []

    dop_seasonal = max(m, n) - 3
    # print('dop_seasonal = ', dop_seasonal)
    ss_value = [0] * (dop_seasonal + 1)

    indicator = m * s + n
    years = YearsCounter(train_mas)

    for i in range(indicator, len(train_mas)):
```

```
            y_values.append(train_mas[i])
            ns = []
            ss = []
            ms = []
            mas_mini = []
            Y_st = 0
            Y_s0 = 0
            for j in range(1, n + 1):
                ns_value = train_mas[i - j]
                mas_mini.append(int(ns_value))
            # print('First mas_mini = ', mas_mini)
            for l in range(dop_seasonal + 1):
                for j in range(1, n + 1):
                    ss_value[l] = train_mas[i - l * s - j]
                    mas_mini.append(int(ss_value[l]))
                # print('Second mas_mini = ', mas_mini)
            # for j in range(1, n + 1):
            # ss_value = train_mas[i - s - j]
            # mas_mini.append(int(ss_value))
            # print('Second mas_mini = ', mas_mini)
            for j in range(1, n + 1):
                ms_value = train_mas[i - m * s - j]
                mas_mini.append(int(ms_value))

            for j in range(1, years + 1):
                if j * i <= len(train_mas) - 1:
                    Y_st = Y_st + train_mas[j * i]
            Y_st = Y_st / years

            for j in range(1, years + 1):
                for k in range(1, s):
                    if k * j <= len(train_mas) - 1:
                        Y_s0 = Y_s0 + train_mas[k * j]
            Y_s0 = Y_s0 / (years * s)

            U_t = Y_st / Y_s0
            # print(U_t)
            mas_mini.append(float(U_t))
            # print('Third mas_mini = ', mas_mini)
            mas_large.append(mas_mini)

    return mas_large, y_values

xTrain, yTrain = NonStatSeries(newDataTrain, n, m, s)
```

```
xTest, yTest = NonStatSeries(newDataTest, n, m, s)

node_layers = [len(xTrain[0]), 250, 200, 1]
lr = 0.15
epochs = 100


def NeuralNetwork(masTrain, epochs, nodes, lr, n):
    weights = TheInitializeWeights(nodes)
    # print('NETWORK weights = ', weights)
    xTrain, yTrain = StatNonStatSeries(newDataTrain, n)

    for epoch in range(1, epochs + 1):

        # print('EPOCH:', epoch)
        weights = Train(xTrain, yTrain, n, lr=lr,
weights=weights)
        # print('NETWORK TRAIN weights = ', weights)
        # print("Training
Accurancy:{}".format(Accurancy(xTrain, yTrain, weights)))

        if (epoch % 20 == 0):
            print("Epoch {}".format(epoch))
            print("Training
Accurancy:{}".format(Accurancy(xTrain, yTrain, weights)))

    # print('Finishing Train!')

    return weights


def TheInitializeWeights(nodes):
    layers = len(nodes)

    # print('InitializeWeights: layers = ', layers)
    weights = []
    for i in range(1, layers):
        we = [[np.random.uniform(-1, 1) for p in
range(nodes[i - 1])] for r in range(nodes[i])]
        weights.append(np.matrix(we))

        # w = [[np.random.random for k in range(nodes[i -
1] + 1)] for j in range(nodes[i])]
        # weights.append(np.matrix(w))
```

```
    return weights

def TheForwardPropagation(x, weights, layers):
    activations = [x]
    layer_input = x
    for j in range(layers):
        #print('FORWARD iteration = ', j)
        activation = LeakyRelu(np.dot(layer_input,
weights[j].T))
        #print('FORWARD activation = ', activation)

        activations.append(np.matrix(activation))

        layer_input = activation # Augment with bias

    #print('FORWARD activations = ', activations)
    return activations

def LeakyRelu(x):
    x = x.A1
    data = [max(0.05*value,value) for value in x]
    return np.array(data, dtype=float)

def LeakyReluDerivative(x):
    x = x.A1
    data = [1 if value>0 else 0.05 for value in x]
    return np.array(data, dtype=float)


def BackPropagation1(y, activations, weights1, layers):
    # print('BACK activations = ', activations)
    outputFinal1 = activations[-1]
    # print('BACK outputFinal = ', outputFinal)
    error = np.matrix(y - outputFinal1[0])
    # print('len_layers = ', layers)

    newActivations = activations

    for i in range(0, layers, -1):
        # print('BackPropagation: ITERATION = ', i)
        currActivation = activations[i]
        if i == layers:
            newActivations[-1] = 555
        if (i == (len(layers) - 1)):
            prevActivation = activations[i - 1]
```

```
            else:
                prevActivation = activations[0]

            delta = np.multiply(error,
LeakyReluDerivative(currActivation))

            for j in range(len(weights1)):
                weights1[i - 1][j] = lr * np.multiply(delta.T,
prevActivation)
            w1 = np.delete(weights1[j - 1], [0], axis=1)

            error = np.dot(delta, w1)

        # print('BACK newActivations = ', newActivations)
        # print('BACK weights = ', weights)
        return weights1


def Train(xTrain, yTrain, n, lr, weights):
    layers = len(weights)

    for i in range(len(xTrain)):
        # print('TRAIN ITERATION:', i)
        x = xTrain[i]
        # print('TRAIN XxTrain = ', x)
        y = yTrain[i]
        x = x
        # print('TRAIN XxTrain2 = ', x)
        # print('x append (1, x) = ', x)

        activations = TheForwardPropagation(x, weights,
layers)
        # print('Train: activations = ', activations)
        weights = BackPropagation1(y, activations,
weights, layers)
    return weights


def Predict(item, weights):
    # print('Predict item1:', item)
    layers = len(weights)
    # print('Predict layers:', layers)
    item = item
    # print('Predict item:', item)
```

```
    activations = TheForwardPropagation(item, weights,
layers)

    # print('Predict activations:', activations)
    # print('Predict output=', activations[-1])
    outputFinal1 = activations[-1].A1
    # print('Predict outputFinal:', outputFinal)

    return outputFinal1


def Accurancy(xTrain, yTrain, weights):
    sumFP = 0
    sumF = 0
    xx1 = []
    for i in range(len(xTrain)):
        x = xTrain[i]
        y = yTrain[i]
        xx1.append(i)
        output = Predict(x, weights)
        # print('ACCURANCY guess = ', output)
        sumFP = sumFP + abs(y[0] - output)
        sumF = sumF + y[0]
        acc = (output * 100) / y[0]

        wape = 100 - ((sumFP / sumF) * 100)

    fig2 = plt.figure()
    ax2 = fig2.add_subplot()
    ax2.plot(xx1[0:80], output[0:80], label="Спрогнозовані
значення")
    ax2.plot(xx1[0:80], yTest[0:80], label='Вихідні
значення', color='green')
    ax2.legend()
    ax2.set_xlabel('Час, год.')
    ax2.set_ylabel('Енергоспоживання, МВт*год.')
    plt.show()

    return wape

weights = NeuralNetwork(newDataTrain, epochs = epochs,
nodes = node_layers, lr = lr, n=n)

print("Testing Accuracy: {}".format(Accurancy(xTest,
yTest, weights)))
```