ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

НЕЙРОННІ МЕРЕЖІ В СИСТЕМАХ АВТОМАТИЗАЦІЇ

курс лекцій з навчальної дисципліни

«Нейронні мережи в системах автоматизації»

(за освітньо-професійною програмою другого (магістерського) рівня освіти «Інтелектуальні системи управління та робототехнічні комплекси в гірничо-металургійному виробництві» спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»)

Рекомендовано Науково-методичною радою
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 6 від «24» травня 2024 р.)
Обов'язково до розміщення в репозиторії

Запоріжжя 2024

Нейронні мережі в системах автоматизації: курс лекцій з дисципліни «Нейронні мережі в системах автоматизації» (за освітньо-професійною програмою другого (магістерського) рівня освіти «Інтелектуальні системи управління та робототехнічні комплекси в гірничо-металургійному виробництві» спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка») / Уклад. О.В. Разживін. Запоріжжя: ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2024. 293 с.

Конспект лекцій присвячено роботі інтелектуальний систем управління в реальному часі з метою вироблення керуючих рішень у темпі надходження нової інформації, тому обчислювальні ресурси традиційних комп'ютерів швидко виявляються вичерпаними, якщо реалізується досить складна ШНС. Така ситуація характерна для систем аналізу та стиснення зображень, при управлінні технологічними процесами та рухомими об'єктами. У зв'язку з цим велике практичне значення набувають питання використання спеціалізованих обчислювальних засобів, що допускають паралельну обробку інформації.

Рекомендовано для здобувачів другого (магістерського) рівня освіти спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка»усіх форм навчання другого (магістерського) рівня вищої освіти).

Самостійне електронне текстове мережеве видання

Затверджено на засіданні кафедри
автоматизації, електро- та робототехнічних систем
Протокол № 8 від «30» квітня 2024 р.

Узгоджено:
Секретар Редакційної ради

_____ Малій Х. В.
«17» травня 2024 р.

ЗМІСТ

ВСТУП

Поняття «штучні нейронні мережі» оформилося в 1940-і роки завдяки основній роботі У. Мак-Каллока і Ч. Піттса [1], в якій було запропоновано модель мозку як множини нейронів, що мають однакову структуру.

Термін "штучний інтелект" був введений в результаті невдалого перекладу з англійської "artificial intelligence", що насправді означає "уміння міркувати розумно".

Родоначальником штучного інтелекту вважають [1] середньовічного іспанського філософа, математика та поета Раймонда Луллія, який намагався у XIII столітті створити механічну машину для вирішення різних завдань на основі розробленої ним класифікації понять, інакше кажучи, структури деякої бази знань. У XVIII столітті ідею Луллія продовжили Лейбніц і Декарт, які запропонували універсальні мови класифікації – перші теоретичні роботи у сфері штучного інтелекту. Проте, як наука, штучний інтелект сформувався внаслідок досліджень Норберта Вінера – батька кібернетики. В даний час – це одна з найбільш перспективних та престижних областей інформатики..

1 ОСНОВИ ТЕОРІЇ НЕЙРОННИХ МЕРЕЖ

“Все життя – від найпростіших до найскладніших організмів, включаючи людину, є довгий ряд ускладнених врівноважень навколишнього середовища.”.

І.П. Павлов

1.1. Деякі відомості про мозок людини

Інтелектуальні здібності людини нерозривно пов'язані з діяльністю мозку, але так не завжди вважали. Стародавні єгиптяни вважали, що вмістищем розуму є не мозок, а серце людини. Мозок давньоєгипетських мумій безжально видалявся з черепа.

Механізми мисленнєвої діяльності людини багато в чому ще залишаються не пізнаними, проте вивчення безперервно продовжується.

Нормальна маса мозку людини становить від 1000 до 2000 р. Так, мозок великого письменника І. З. Тургенєва важив близько 2000 р., а мозок іншого великого письменника – А. Франса – лише 1100 р.

Було встановлено, що у німців середня маса мозку становить 1291 г, у швейцарців – 1374 г, у росіян та українців – по 1377 г, а у бурятів – 1508 р. Цікаво, що середній розмір мозку неандертальця кілька перевищував середній розмір мозку сучасної людини і досягав 1610 року.

Виявлено також, що середній розмір мозку стародавніх єгиптян протягом 18 династій фараонів зменшувався на 1 грам кожні 10 років, що за 3000 років склало близько 300 г.

У порівнянні з іншими ссавцями, мозок яких подібний до людського, людина хоч і виділяється розмірами головного мозку, але не є чемпіоном. Так, мозок синього кита важить до 6800 г, а мозок слона – близько 5000 р. Втім, питома маса мозку людини становить 1/50 стосовно маси тіла, тоді як в кита цей показник лише 1/10000.

Таким чином, між розміром мозку людини і рівнем його інтелекту немає прямої залежності, важливіше її внутрішню будову та особливості функціонування. Від тварин людина відрізняється, мабуть, лише тим, що він краще розвинені окремі ділянки мозку. Непрямим свідченням цього є той факт, що 97% генному людини та шимпанзе збігаються. У 3% відмінностей укладаються всі зовнішні ознаки людини - будова руки, стопи, черепа і т. д., а також внутрішні ознаки - здатність до членороздільного мовлення, абстрактного та логічного мислення.

Головний мозок людини є дуже складною структурою, що містить безліч відділів і шарів.

Мозок складається з сірої та білої речовини. Сіра речовина є

мережею дендритів, аксонів і тіл нервових клітин. Міє-лінізовані (ізольовані) волокна, що з'єднують різні області мозку один з одним, з органами почуттів та м'язами, утворюють білу речовину.

Мозок складається з двох півкуль, кожна з яких у свою чергу включає лобну, тім'яну, скроневу і потиличну частини.

У лобової частини знаходяться відділ емоцій і центри управління рухами: права півкуля відповідає за рух лівої руки та ноги, а ліве – за рух правої руки та ноги), у тім'яній частині – зона тілесних відчуттів та дотиків. До неї примикає скронева зона, в якій розташовані центр мови, слуху, смаку. У потиличній частині знаходиться зоровий відділ.

У мозку існують структурно відокремлені відділи, такі як кора, гіпокамп, таламус, мозок, мигдалина, смугасте тіло і т. д. (див. рис. 1.1).

Нижчі відділи мозку відповідають за базові функції: довгастий мозок контролює дихання, травлення та серцебиття, мозок управляє органами руху і т.д.

Найпізніший відділ головного мозку - це кора, або неокортекс. На відміну від багатьох ссавців, у людини кора становить значну частину мозку. Вона є досить тонким шаром (2–3 мм) нервової тканини, площа якого становить приблизно 2500 см^2. Ця поверхня «скомкана» в черепній коробці і утворює численні звивини та борозни.

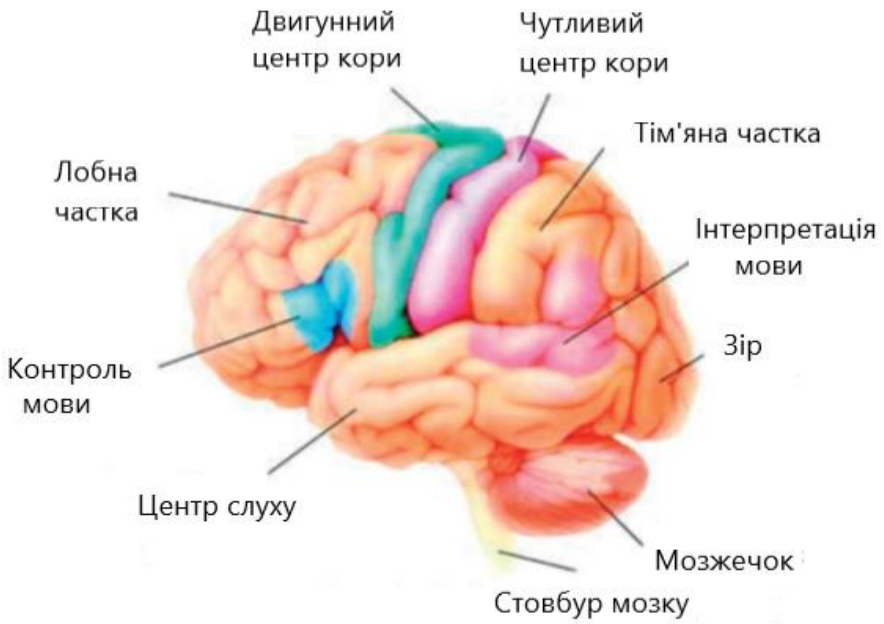


Рисунок 1.1 - Карта головного мозку

Лобові частки мозку мають особливе значення: вважається, що вони відповідають прийняття рішень, і навіть за особисті якості людини. Префронтальна кора мозку людини (передня частина лобових часток) займає 29% усієї кори, тоді як у шимпанзе – лише 17%.

Кора мозку містить від 10 до 20 млрд нейронів (за іншими оцінками – до 100 млрд). Кожен нейрон (нервова клітина) може мати зв'язку

приблизно з 10 000 інших нейронів. На забезпечення роботи мозку припадає приблизно 1/4 всіх енергетичних витрат організму та споживається 1/5 кисню тіла.

Нейрони мозку в процесі життя людини поступово відмирають, але це слабко відбивається на його функціонуванні, оскільки людина використовує, як вважають, не більше 10% об'єму мозку, так що зменшення числа нейронів у мозку похилого віку від 100 до 70% може і не бути особливо важливим. Обробка величезних обсягів інформації мозком здійснюється дуже швидко, за частки секунди, незважаючи на те, що сам нейрон є повільно чинним елементом з часом реакції кілька мілісекунд.

Мозок відокремлений від решти тіла спеціальною системою, що фільтрує (гематоенцефалічний бар'єр), яка захищає його від проникнення шкідливих речовин. Захист забезпечується низькою проникністю кровоносних судин мозку, а також гліальними клітинами, що оточують нейрони. Однак деякі речовини (пари бензину, наркотики тощо) легко долають цей бар'єр і можуть викликати незворотну деградацію кори головного мозку.

Якщо в людини домінує ліва півкуля мозку, то вона – «правша», якщо права – то вона шульга. Ліва півкуля відповідає за абстрактне логічне мислення, праве – за образне. Відомо, що у дітей молодшого шкільного віку домінує правопівкульне (образне) сприйняття, а приблизно до 14 років може відбутися перехід до лівопівкульного (логічного) домінування.

Зв'язок між півкулями здійснює мозолисте тіло мозку.

Інформація про функціонування півкуль була отримана при спостереженні хворих з так званим «розщепленим мозком», у яких розсічено мозолисте тіло (дана операція виконується при лікуванні епілепсії). У цих людей інформація від кожного ока обробляється лише однією півкулею: від лівого ока – правим, від правого – лівим. Якщо такій людині запропонувати розглядати різні предмети тільки лівим оком, то вона не зможе їх назвати, оскільки права півкуля не відповідає за мовні функції, вона здатна обробляти тільки найпростіші мовні команди. Відповідно, якщо демонструвати предмети правому оку, то виникають труднощі з їх переміщенням. Цікаво, що хворі з «розщепленим мозком» можуть грати в шахи як два різні гравці, спостерігаючи дошку правим і лівим оком по черзі, так що один «гравець» не знає про задуми іншого.

У той самий час одна півкуля може приймати він функції другого. Відомі випадки, коли люди, які втратили одну з півкуль мозку, поступово відновлювали практично всі втрачені функції (наприклад, мова).

Досить добре вивчено спеціалізацію окремих ділянок головного мозку. Так, у скроневій його частці знаходяться первинні слухові, а в потиличній – первинні зорові поля.

Зір дає людині найбільший обсяг інформації. Сітківка людського ока містить близько 110 млн світлочутливих клітин. Око з'єднане з

мозком 10 млн нервових волокон, які пов'язані з зорової корою головного мозку.

Центральна нервова система складається з головного та спинного мозку.

Спинний мозок відповідає в основному за прийом сенсорних сигналів від тіла до мозку та за передачу нервових сигналів від мозку до м'язів. Швидкість поширення збудження вимірюється десятками метрів на секунду.

Пройшовши шлях через нейронні структури мозку від рецепторів (слухових, зорових та інших) до виконавчих органів, вхідна інформація перетворюється на набір керуючих впливів, адекватних ситуації.

За місцем розташування та функцій у нервовій системі розрізняють сенсорні (рецепторні), вставні (інтернейрони) та ефективні (мотонейрони) нейрони.

Рецепторні нейрони сприймають енергетичні впливи та сигнали середовища. Взаємодіючі один з одним інтернейрони здійснюють внутрішню обробку інформації, а мотонейрони передають результати цієї обробки безпосередньо на виконавчі системи організму, в якості яких виступають м'язи та залози внутрішньої секреції.

Нервові волокна, що передають головний мозок збудження з його периферичних відділів і спинного мозку, називаються аферентними (висхідними) шляхами. Порушення, спрямоване з мозку, називається еферентними (низхідними) шляхами.

Окремі нейрони, з'єднуючись між собою, набувають нової якості, яка в залежності від характеру міжнейронних сполук має різний рівень біологічного моделювання:

- група нейронів;
- нейронна мережа;
- нервова система;
- розумова діяльність;
- мозок.

Протягом життя людини кожна молекула тіла багаторазово оновлюється (з періодом приблизно сім років), тим часом пам'ять зберігає події життя людини протягом багатьох років.

Вважається, що здатність до забування необхідна для нормальної життєдіяльності, оскільки вона пов'язана зі здатністю до узагальнення та накопичення досвіду. Деякі люди практично позбавлені здатності до забування. Таке явище називається ейдетізмом, або фотографічною пам'яттю.

Пам'ять людини ділиться на довготривалу та короткочасну.

У довгостроковій пам'яті зберігаються об'єкти та зв'язки між ними, тобто символьна інформація. Час отримання інформації з довгострокової пам'яті становить зазвичай до 70 мс (за цей час людина дізнається той чи інший образ або ситуацію і реагує відповідним чином). Людина здатна розпізнавати близько 20 образів на секунду.

У короткочасній пам'яті знаходяться дані, одержувані за допомогою органів чуття.

Переміщення даних з короткочасної пам'яті в довготривалу займає 15-20 хв. При цьому запис одного образу та встановлення зв'язків між ним та іншими записами довгострокової пам'яті вимагає близько 7 с.

Людина ефективно сприймає і запам'ятовує близько 7 ± 2 градацій властивостей будь-якого об'єкта (саме тому ми вважаємо, що у веселки сім кольорів). Це саме стосується і наборів фактів і зв'язків між ними, які витягуються як єдине ціле. У англомовній літературі такі набори називають чанками. Фахівець у конкретній предметній області пам'ятає від 50000 до 100000 чанків. Накопичення такого обсягу знань займає від 10 до 20 років.

Як же відображається чанк у мозку людини? Відповідно до домінуючими уявленнями про роботу мозку кожному чанку відповідає група нейронів, між якими сформовані стійкі зв'язки. Інакше кажучи, ці нейрони порушуються одночасно, реагуючи на певну вхідну інформацію.

У мозку людини щодня виникає приблизно 50 000 думок.

При вирішенні інтелектуальних завдань людина використовує три свої основні здібності:

• до перебору,
• до абстракції,
• до математичної індукції.

Здатність людини до перебору пов'язана з можливістю послідовного перемикання уваги з одного предмета на інший з впізнанням предмета, що шукається. Ця здатність вельми обмежена - в середньому людина може впевнено (не збиваючись) перебирати в межах 1000 предметів (елементів). Засобом подолання цієї обмеженості є його здатність до абстракції, завдяки якій людина може поєднувати різні предмети або екземпляри в одне поняття, замінювати безліч елементів одним елементом (іншого роду). Здатність людини до математичної індукції дозволяє йому впоратися з нескінченними послідовностями.

Ще одна важлива властивість людської пам'яті полягає у її асоціативності. Людина може відновлювати правильний образ за її спотвореною копією, або, що те саме, відтворювати повну інформацію щодо її частини.

Обмеженість здатності людини до перебору дозволяє запровадити поняття простий і складної системи. Під простий розумітимемо таку систему, в якій людина може впевнено перебрати всі шляхи взаємодії між її елементами, а під складною – таку, в якій він цього зробити не в змозі. Між простими та складними системами немає чіткої межі, але умовно можна вважати, що проста система має не більше 7±2 елементів.

1.2 Проблеми теорії та практики формування знань

1.2.1 Сучасні уявлення про штучний інтелект

Сучасна ідея штучного інтелекту формуються наступним чином: "Єдиний об'єкт, здатний мислити, - це людський мозок. Тому будь-який мислячий пристрій має якимось чином відтворювати його структуру" [1]. У зв'язку з цим було прийнято орієнтування на програмно-апаратне моделювання структури мозку та створення елементів, аналогічним нейронам – нервовим клітинам мозку.

Кожен нейрон реалізує деяку функцію над вхідними значеннями. Якщо значення функції перевищує певну величину – поріг, то нейрон порушується і формує вихідний сигнал передачі його іншим нейронам. Мозок отримує вхідну інформацію від рецепторів (слухових, зорових та інших), потім вона обробляється нейронними структурами, перетворюється на набір впливів, що управляють на організм.

Діяльність [1] було показано, що мережі, які з штучних нейронів, здатні, у принципі, обчислити будь-яку арифметичну чи логічну функцію. Автори запропонували використовувати штучні нейронні мережі (ІНП), елементами яких є штучні нейрони, виконані на бінарних порогових перетворювачах та функціонуючі за принципом «усі чи нічого». Такі мережі виявилися здатними навчатися розпізнаванню образів і узагальнення інформації, тобто володіли якостями, властивими живому мозку.

У 1949 р. Д. Хебб [2] припустив, що умовний рефлекс, відкритий І. П. Павловим, виникає внаслідок можливості окремих нейронів до встановлення асоціацій, і сформулював відповідне правдоподібне правило навчання біологічних нейронів.

Перше практичне використання ШНМ посідає кінець 1950-х років, воно пов'язані з винаходом Ф. Розенблаттом персептрона [3]. Основна ідея Розенблатта зводилася до заміни жорстких логічних схем Мак-Каллока та Піттса системами зі статистичними властивостями. Персептрон продемонстрував здатність ШНМ до навчання та вирішення завдань класифікації. Цей успіх викликав сплеск інтересу до дослідження таких систем.

Приблизно водночас Б. Відров і М. Є. Хофф [4] запропонували інший навчальний алгоритм налаштування адаптивних лінійних нейронних мереж. Однак виявилося, що одношарові мережі Розенблатта та Відрова мають подібні обмеження, що звужують сферу їх застосування. Це було доведено у роботі М. Мінського та С. Пайперта [5]. Автори сформулювали та довели ряд теорем, що підтверджують принципову обмеженість одношарових.

ШНМ та їх нездатність вирішувати багато простих завдань, у тому числі реалізувати функцію «що виключає АБО».

Після появи цієї книжки Розенблатт і Видров розробили

багатошарові мережі, вільні від виявлених недоліків. Однак їм не вдалося модернізувати свої навчальні алгоритми так, щоб ці складніші мережі можна було налаштовувати автоматично.

Ці результати викликали деяке розчарування у можливостях ШНМ, яке тривало до початку 80-х років, хоча в 70-ті роки були опубліковані важливі роботи з таких мереж Т. Кохонена [6] та С. Гроссберга [7].

У 1980-ті роки з'явилися потужні персональні комп'ютери та робочі станції, що дозволило виконувати складні експерименти з ШНМ.

У 1982 р. Дж. Хопфілд успішно застосував методи механіки для опису роботи одношарових повнозв'язкових динамічних ШНМ [8]. Кохонен запропонував новий клас штучних нейронних мереж для вирішення задач векторної класифікації [9].

Проте найбільше значення відродження інтересу до ШНМ мало поява алгоритму *зворотного поширення* помилки, що дозволяє навчати багатошарові ШНМ прямого поширення [10].

Алгоритм зворотного поширення помилки (існуючий у численних модифікаціях) показав дуже хороші результати щодо навчання ШНМ при вирішенні багатьох прикладних завдань, пов'язаних з розпізнаванням тексту, символів тощо. Разом з тим цей алгоритм є локальним, тобто гарантовано працює тільки тоді , коли функція помилки, що мінімізується при навчанні, є унімодальною. У багатьох завданнях функція помилки мультимодальна, тому в останні роки поряд з алгоритмом зворотного поширення для навчання ШНМ використовуються такі універсальні алгоритми глобальної оптимізації, як алгоритм відпалу металу [11], генетичний алгоритм, «роєвий інтелект», метод мурашиних колоній [11].

Детальний опис історії розвитку ШНМ можна знайти в [7-11]. Чимало робіт з теорії та практики застосування ШНМ ([7–11] та ін.).

В результаті можна виділити такі напрямки використання штучних нейронних мереж:

• *класифікація образів.* Завдання полягає у вказівці належності вхідного образу, представленого вектором ознак, до одного або кількох попередньо визначених класів. До подібних завдань відносяться розпізнавання символів, мовлення, класифікація електрокардіограм, клітин крові тощо.;

• *апроксимація функцій*. Припустимо, що є навчальна вибірка, задана парами вхід-вихід: $((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$, отриманими від системи, що описується невідомою функцією *f*. Завдання апроксимації полягає у знаходженні такої ШНМ, поведінка якої відповідає даній функції;

• *кластеризація*. При розв'язанні задачі кластеризації опис класів наперед не відомий. Є безліч об'єктів, яке потрібно розбити на групи близькоспоріднених елементів (кластери). Кластеризація

застосовується для обробки зображень, отримання знань, стиснення даних;

• *прогнозування*. Нехай задані n дискретних відліків {$y(t_1)$, $y(t_2)$, ..., $y(t_n)$} у послідовні моменти часу t_1, t_2, ..., t_n. Завдання полягає у передбаченні значення $y(t_{n+1})$. Прогнозування має велике значення при прийнятті рішень у різних галузях людської діяльності;

• *асоціативна пам'ять.* У сучасних обчислювальних машинах звернення до пам'яті доступне шляхом вказівки адреси, яка залежить від її змісту. Асоціативна пам'ять повинна бути доступна за вказівкою заданого змісту, який може бути спотвореним або неповним. Така властивість характерна для людської пам'яті, коли невелика деталь об'єкта дозволяє уявити його повний опис. Реалізація асоціативної пам'яті особливо важлива при створенні мультимедійних додатків та баз знань;

• *оптимізація*. Багато проблем у науці, техніці, медицині та економіці можуть розглядатися як завдання оптимізації. Під завданням оптимізації розуміється знаходження такого рішення, яке задовольняє системі обмежень та забезпечує екстремум заданої цільової функції.;

• *управління динамічною системою* передбачає переведення її з деякого початкового стану на задане цільове. Використання ШНМ здатне збагатити апарат класичної теорії управління.

Ці загальні напрями у різних поєднаннях знаходять використання у практичних додатках, яких ставляться обробка зображень, управління антропоморфними роботами і маніпуляторами, техніка зв'язку, управління фінансовими ринками, активна реклама в Інтернеті тощо.

Нейронні мережі можуть бути ефективно використані для вирішення завдань обробки даних техніки, в яких є елементи комбінаторної оптимізації та розпізнавання образів. До них відносяться [11]:

- адаптивне управління в умовах непередбачених змін динамічної схеми та апріорної невизначеності моментів, що обурюють;
- мінімізація втрат системи електропостачання (СЕС) внаслідок оптимального струморозподілу;
- діагностування обладнання на основі автономного аналізу формованої на телеметричній інформації;
- підвищення якості функціонування системи керування шляхом оптимізації складу селективних ознак та оперативного визначення параметрів відносного стану.
- у промисловому виробництві для управління процесами з невизначеними взаємодіями об'єктів, що беруть участь у процесі, а також для діагностики роботи машин, оцінки якості продукції та ін.;
- при передачі інформації для стиснення та відображення даних, фільтрації сигналів, розпізнавання мовлення, перетворення текстової інформації та ін;
- в електроніці для побудови систем технічного зору,

розміщення мікросхем при проектуванні плати, для нелінійного моделювання та інших.;

- у фінансовій справі для автоматичного читання документів, прогнозування вартості валюти, проведення загального фінансового аналізу;
- в автомобілебудуванні, космонавтиці та аеронавтиці для проектування складних систем управління, моделювання (імітації) процесів руху, побудови алгоритмів управління рухом.

Цей перелік не охоплює всієї різноманітності розв'язуваних завдань. В даний час нейронні моделі застосовують практично у всіх видах діяльності.

Існує безліч пакетів прикладних програм для роботи з нейронними мережами, таких як NeuroSolutions фірми NeuroDimension Inc., NeuralWorks Professional U/Plus з модулем UDND фірми Neural Ware, Inc., Process Advisor фірми AIWare, Inc., NeuroShell2 фірми Ward Systems Group, BrainMaker Pro фірми California Scientific Software і т.д.

У конспекті використовуються приклади роботи з нейромережами, орієнтовані використання системи MatLab Simulink з розширенням Neural Network toolbox. Це зумовлено низкою причин:

- пакет Neural Network toolbox дозволяє легко вбудовувати ШНМ у моделі, отримані за допомогою інших компонентів MatLab;
- система MatLab відрізняється відкритістю, що дозволяє легко створювати нові та модифікувати існуючі програмні модулі.

Вивчення принципів використання штучних нейронних мереж є необхідним компонентом підготовки магістрів за напрямком «Автоматизація та комп'ютерно-інтегровані технології».

1.2.2 Особливості та проблеми формування знань

Прагнення вчених та фахівців у галузі автоматичного управління зводяться до того, щоб побудувати системи, які могли б приймати рішення та виробляти управління на основі відомостей про навколишнє середовище, а також з урахуванням досвіду та мотивацій. Для цього система повинна мати необхідні знання.

Знання зазвичай поділяються на поверхневі та глибинні знання.

Поверхневі знання – це знання про видимі взаємозв'язки між подіями та фактами у предметній області, наприклад, у поданні електричного дзвінка: "якщо натиснути на кнопку, то дзвінок задзвенить".

Глибинні знання – це абстракції, аналогії, схеми, що відбивають природу процесів, які у предметної області, наприклад, знання принципової схеми дзвінка і його роботи.

За категоріями знання поділяються на:

– концептуальні (понятійні) - це знання, що відображають розумові здібності;

– фактуальні (предметні) – це відомості про якісні та кількісні характеристики, тобто інформація та дані;

– алгоритмічні (процедурні) – це знання про технології, методи та програми, орієнтовані на конкретну сферу застосування.

З цього переліку видно, що концептуальні знання є основою для побудови інших категорій знань.

У звичайних системах автоматизації функції діяльності реалізуються на синтаксичному (формальному) рівні, що визначається інформацією, даними та алгоритмом управління процесом. У цьому поведінка системи визначено апріорно, а зміни виробничого середовища враховуються у межах при створенні алгоритму. Модель зовнішнього світу (виробничого середовища) у пам'яті системи відсутня [2].

Для інтелектуалізації процесу управління функції управління мають задаватися на семантико-прагматичному рівні, що дозволяє побудувати процес управління не так на основі аналізу бази даних, але в основі аналізу бази знань. При цьому в системі управління створюється новий стиль подання інформації, у тому числі й моделі зовнішнього світу, при зміні якої формується найбільш правильна відповідь.

У вирішенні будь-якої задачі вихідні дані та знання поетапно *трансформуються* [1].

Дані (факти), які характеризують об'єкти, процеси та явища предметної області, а також їх властивості, трансформуються в наступній послідовності:

1. D1 – результати вимірювань та спостережень;
2. D2 – матеріальні носії інформації (таблиці, протоколи, довідники);
3. D3 – моделі (структури, графіки тощо);
4. D4 - дані в комп'ютері мовою опису даних;
5. D5 – бази даних у комп'ютері.

Знання – це метадані, які є закономірністю предметної області, зазвичай засновані на даних теоретичних досліджень та даних, отриманих емпіричним шляхом. Інакше кажучи, знання – це продукт розумової діяльності. При обробці на ЕОМ знання трансформуються аналогічно даним у такій послідовності:

1. Z1 – знання у пам'яті людини, як результат її мислення;
2. Z2 – матеріальні носії знань, під якими розуміють підручники, методичні посібники, інструкції, керівні матеріали тощо.);
3. Z3 – поле знань, як умовний опис чи модель об'єктів предметної області, необхідне переходу до формального уявлення знань;
4. Z4 – знання, описані мовами їх уявлення, тобто семантичні конструкції та мережі, фрейми, формальні логічні моделі;

Наведені тут терміни мають такі визначення:

– семантична конструкція – це смислове позначення поняття;

– семантична мережа – це орієнтований граф, вершини якого – поняття, а дуги – відносини між поняттями;

– кадр (каркас, рамка) – це структура знань, абстрактний образ, який представляється деяким стереотипом.

5. Z5 – база знань на машинних носіях інформації як основа інтелектуальної системи.

При спробі формалізувати людські знання виникає проблема, пов'язана з відсутністю апарату для їхнього опису. Крім цього, деякі описи не можуть бути інтерпретовані певними знаннями. У зв'язку з цим виникає проблема "нечітких знань". Для вирішення такої проблеми американським математиком Лотфі Заде було запропоновано апарат нечіткої (fuzzy) логіки.

Центральна парадигма інтелектуальних технологій сьогодні – це опрацювання знань. Тому *інтелектуальними системами* називають системи, в яких ядро складають *база знань чи модель предметної галузі, описана мовою високого рівня, наближеного до природного*.

2. НЕЙРОННІ МЕРЕЖІ. БАЗОВІ ПОНЯТТЯ

2.1 Принципи функціонування біологічного нейрона

Інформацію про довкілля та внутрішній стан людина отримує від сенсорної системи (аналізатора) – частини нервової системи, що складається з периферичних рецепторів (органів почуттів), які пов'язані з центральною нервовою системою нервовими волокнами.

У сенсорних органах енергія впливу перетворюється на рецепторний потенціал, який трансформується в імпульсну активність нервової клітини – потенціал дії. Цей потенціал подається до сенсорного центру, клітини якого перетворюють (перекодують) нервовий сигнал. Сенсорні центри пов'язані з руховими та асоціативними відділами мозку. Ідентифікуючи властивості сигналів, вони подають у рухові центри імпульси, які викликають збудження чи гальмування (дія чи бездіяльність).

Центральною ланкою в цій системі є мозок, що складається з нервових клітин – нейронів, кількість яких перевищує 100 мільярдів і кожен із яких має в середньому 10 тис. зв'язків з іншими нейронами.

Біологічний нейрон, схема якого наведено на рис. 2.1, має тіло – сому, дерево входів – дендрити та вихід – аксон.

Довжина дендритів може досягати одного міліметра, а довжина аксона – сотень міліметрів. Дендрити пов'язані із закінченнями нервових клітин вузлами, які називаються синапсами. Передача імпульсу в синапсі є процес звільнення певної хімічної речовини - нейротрансмітера.

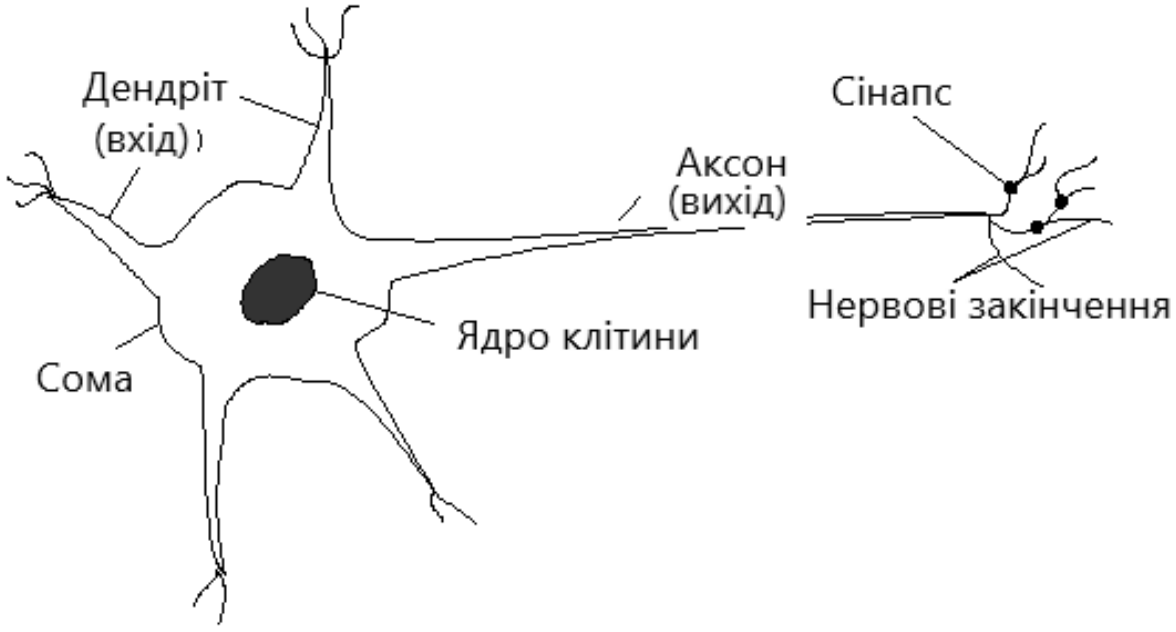


Рисунок 2.1 – Будова біологічного нейрона

Нейротрансмітер дифундує через сінаптичну щілину, збуджуючи або загальмовуючи залежно від типу синапсу здатність нейрона-приймача генерувати електричні імпульси. Синапс може налаштовуватися імпульсами, що проходять через нього, тобто активність синапсу визначається передісторією і діє як пам'ять.

Нейрони взаємодіють за допомогою серії імпульсів протягом кількох мілісекунд. Інформація передається шляхом частотної модуляції імпульсів у діапазоні від одиниць герц до кількох сотень герц. При цьому за одну передачу надсилається кілька біт інформації. Незважаючи на порівняно низьку швидкість та малі обсяги пересилок, складні рішення можуть прийматися швидко у зв'язку з величезним паралелізмом потоків.

Нейрони поділяються на три групи:

- рецепторні, що забезпечують перетворення впливів навколишнього середовища на інформацію для мозку;
- проміжні, що утворюють центральну нервову систему та забезпечують асоціативне перетворення інформації та прийняття рішень;
- ефекторні, що забезпечують зв'язок центральної нервової системи із м'язовою системою.

Усі зв'язки між нейронами фізично паралельні. Один шар клітин проектується на інший, волокна розгалужуються та зливаються, тобто відбувається дивергенція та конвергенція зв'язків.

Практична реалізація такої системи зв'язків представляє поки що серйозну проблему, тому що нейропроцесори, що існують в даний час, мають обмежену систему команд і зв'язків (нейросетьовий базис). Тому практично застосовують спрощені системи, реалізовані або на нейрокомп'ютерах, або шляхом програмного моделювання на звичайних комп'ютерах.

2.2 Поняття нейрокомп’ютера

Розглянуті кількісні показники, що характеризують роботу мозку, не надто вражають. Проте мозок людини поки що якісно перевершує за можливостями сучасні комп'ютери.

Великий вплив на розробку теорії ШНМ справили ідеї коннекціонізму - розділу штучного інтелекту, пов'язаного зі створенням моделей мозку та мисленням людини. З погляду коннекціонізму (від англ. connection – зв'язок), окремі нейрони можна моделювати простими автоматами, а вся складність мозку визначається зв'язками між нейронами. Це забезпечує наступні властивості нейромережевої моделі:

- однорідність системи (елементи нейронної мережі однакові та прості, функція визначається структурою зв'язків);

- надійність системи, побудованої з ненадійних елементів, завдяки надлишковій кількості зв'язків;

- «Голографічність», що забезпечує збереження властивостей системи при руйнуванні її частини.

У завданнях розпізнавання, наприклад, незважаючи на низьку швидкість виконання операцій окремим нейроном (мілісекунди), яка істотно менша за тактову частоту процесорів сучасних комп'ютерів, мозок легко виграє у даних пристроїв. Це є наслідком високого ступеня паралелізму, що забезпечується величезною кількістю паралельно функціонуючих нейронів та міжнейронних сполук, що визначає високу швидкість роботи мозку в цілому.

Результати порівняння характеристик комп'ютера та мозку людини у першому наближенні наведено у табл. 2.1.

Таблиця 2.1 – Якісне порівняння типів обчислювачів

Параметр	*Комп'ютер*	*Мозок*
Процесорний елемент	Складний, високошвидкісний, мала кількість	Простий, низькошвидкісний, велика кількість
Пам'ять	Відокремлена від процесора, локалізована, не асоціативна	Інтегрована з процесором розподілена, асоціативна
Обчислення	Централізовані, послідовні, строго детерміновані	Розподілені, паралельні, адаптивні
Надійність	Сильна чутливість до збоїв	Робастність (живучість)
Спосіб вирішення задач	Цифрова або символьна обробка	Обробка образів та знань
Область дії	Алгоритмізовані завдання	Необмежена робота в умовах невизначеності

Комп'ютер працюють безпомилково лише за відсутності апаратно-програмних збоїв, дотримуючись жорстко визначеного алгоритму. Мозок орієнтований на обробку якісних даних в умовах невизначеності або неповноти, а також при відмови та пошкодження його ділянок. Це також можна вважати наслідком високого ступеня паралелізму (надмірності) міжнейронних зв'язків.

Необхідно відзначити, що не тільки людський мозок, а й мозок примітивних живих істот має потужні здібності, що забезпечують їхнє виживання. Тому не зовсім правильно стверджувати, що штучна

нейронна мережа моделює мозок людини. Скоріше це модель мозку живої істоти, потенційна потужність якої визначається числом нейронів.

В [28] дано таке визначення ШНМ: *нейронна мережа* - це розподілений паралельний процесор, що складається з елементарних одиниць обробки інформації, що накопичують експериментальні знання і надають їх для подальшої обробки.

Така мережа подібна до мозку за двома ознаками:

- знання надходять у нейронну мережу з навколишнього середовища та використовуються в процесі навчання;
- для накопичення знань застосовуються зв'язки між нейронами, які називаються сінаптичними вагами.

У літературі апаратно-програмні комп'ютерні системи, основою функціонування яких покладено ШНМ, часто називають *нейрокомп'ютерами*. Наукова дисципліна, пов'язана з розробкою та дослідженням методів використання ШНМ у різних практичних областях, називається нейрокомп'ютером.

Термін «нейрокомп'ютер» підкреслює важливе відмінність обчислень в нейронних мережах від обчислень у звичайному комп'ютері, хоча комп'ютер у силу своєї універсальності може бути використаний для моделювання ШНМ.

Традиційний комп'ютер складається з чотирьох основних блоків: центрального процесора (ЦП), що складається з арифметико-логічного пристрою (АЛУ) та пристрою керування, пам'яті, пристроїв введення та виведення. Програма виконується комп'ютером послідовно, команда за командою, слідуючи заздалегідь запропонованому алгоритму (хоча це не зовсім вірно для сучасних багатопроцесорних і мультискалярних ЕОМ).

У нейрокомп'ютері АЛУ реалізовано з урахуванням ШНМ, з якою пов'язаний блок навчання (рис. 2.2).

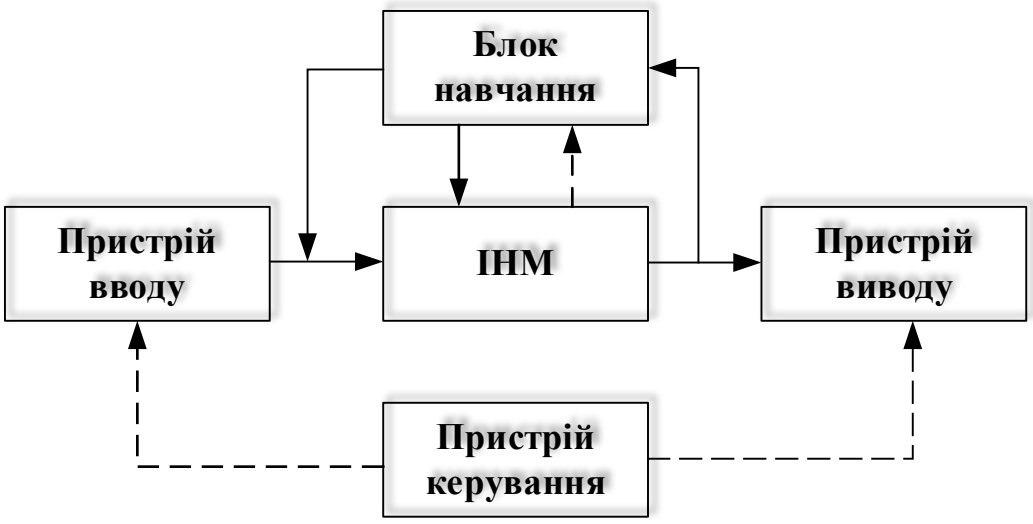


Рисунок 2.2 - Узагальнена структура нейрокомп'ютера

Пам'яттю нейрокомп'ютера можна вважати набір терезів міжнейронних зв’язків, який формується в процесі навчання.

У роботі нейрокомп'ютера принципово присутні два режими: навчання та робітник.

Нейронна мережа має пройти навчання для вирішення конкретного завдання. Завдання навчання полягає в такому налаштуванні коефіцієнтів міжнейронних зв'язків, при якій забезпечується мінімізація помилки уявлення по всій навчальній множині із сукупності навчальних пар, в яких кожному еталонному значенню вхідного образу відповідає бажане (еталонне) значення вихідного образу. З математичної точки зору, процес навчання є рішенням задачі оптимізації.

У робочому режимі блок навчання відключений, і на вхід нейрокомп'ютера подаються довільні сигнали (які не входили або навчальну вибірку). Ці сигнали (вхідні образи) може бути накладений шум. Завдання нейрокомп'ютера полягає у виробленні правильної реакції, що найбільш відповідає його «програмі», під якою можна розуміти топологію ШНМ та набір ваг її міжнейронних зв'язків.

Таким чином, принципова відмінність використання нейрокомп'ютера полягає у відсутності етапу алгоритмізації, який замінюється етапом навчання. Для розуміння переваг, які дає така заміна, слід нагадати поняття формалізованої та неформалізованої задач.

Формалізована задача має алгоритм розв'язання. Прикладом подібних завдань є традиційні обчислювальні завдання: розв'язання алгебраїчних, диференціальних, інтегральних та інших рівнянь, сортування даних тощо. Звичайні ЕОМ орієнтовані саме на задачі, що формалізуються.

Завдання що не формалізується немає описаного алгоритму рішення чи цей алгоритм вимагає надмірних обчислювальних ресурсів.

У процесі розвитку науки і техніки багато завдань можуть переходити з класу, що не формалізуються до класу, що формалізуються, проте завдань, алгоритм вирішення яких не відомий, все ще набагато більше.

Насправді багато завдань можна назвати важко формалізованими, оскільки їм є приватні алгоритми, а універсальний алгоритм не відомий. До цього класу завдань належать такі традиційні завдання штучного інтелекту, як завдання управління складними системами, розпізнавання образів, кластеризації даних, передбачення, апроксимації функцій тощо.

Нейрокомп'ютер є ефективним інструментом для вирішення задач, що важко формалізуються.

2.3 Класифікація нейронних мереж

Найбільш загальна класифікація ШНМ ділить їх у два класу залежно від наявності зворотних зв'язків. Якщо ШНМ немає зворотних зв'язків, вона називається *статичної*, і якщо зворотні існують, то мережа *динамічна* (рекурентна). На рис. 2.3 показані найбільш поширені типи ШНМ, що належать до цих двох класів.

Ще один принцип класифікації ШНМ заснований на їх топології. Відповідно можна виділити повнозв'язкові, багатошарові та слабозв'язані, а також модульні ШНМ.

У *повнозв'язних ШНМ* кожен нейрон передає вихідний сигнал іншим нейронам, у тому числі й самому собі. Усі вхідні сигнали подаються всім нейронам. Вихідними сигналами мережі можуть бути всі або деякі вихідні сигнали нейронів після кількох тактів функціонування мережі. До таких ШНМ належать *мережі Хопфілда.*

Рисунок 2.3 - Класифікація нейронних мереж

У *багатошарових* ШНМ нейрони поєднуються в шари. Шар містить сукупність нейронів із єдиними вхідними сигналами. Число нейронів у шарі може бути будь-яким і не залежить від їхнього числа в інших шарах. Усього мережа складається з N шарів, пронумерованих зліва направо.

У *слабозв'язних* ШНМ нейрони розташовуються у вузлах прямокутної, або гексагональної, решітки. Кожен нейрон пов'язаний із чотирма, шістьма чи вісьмома своїми найближчими сусідами.

У *модульних* (або ядерних) ШНМ, які відносяться до класу мереж прямого розповсюдження, кожен нейрон наступного шару отримує сигнали тільки частини нейронів попереднього. Так виникають нейронні ядра.

За типом структур нейронів ШНМ діляться на *гомогенні* та *гетерогенні*. *Гомогенні* мережі складаються з нейронів одного типу з єдиною функцією активації, а *гетерогенну* мережу входять нейрони з різними функціями активації.

Нейронні мережі можна також поділити на два класи залежно від наявності або відсутності латеральних (від лат. lateralis – бічний) *зв'язків*. Введення в приховані шари ШНМ латеральних (бічних) зв'язків дозволяє моделювати ефекти взаємного послаблення між сусідніми нейронами і посилення власного сигналу нейрона. Це посилює «контрастність» під час вирішення завдань розпізнавання.

Гібридні ШНМ можуть поєднувати в собі ознаки двох, а то й трьох основних видів. Як правило, ці мережі багатошарові, кожен шар яких представляється різною топологією та навчається за певним алгоритмом.

Нарешті, існують *бінарні та аналогові, синхронні або асинхронні* ШНМ.

Вибір топології ШНМ диктується вирішуваним завданням, і навіть досвідом розробника.

При конструюванні ШНМ розробник має, як правило, такі вихідні дані: розмірність вектору вхідного сигналу, розмірність вектору вихідного сигналу, формулювання завдання, що вирішується, вимоги по точності її вирішення.

Крім вибору топології розробник повинен призначити загальну кількість нейронів у мережі та число нейронів за шарами, вид функції активації нейронів, спосіб завдання коефіцієнтів синаптичного зв'язку, метод перевірки працездатності нової мережі.

2.4. Завдання розпізнавання та лінійна машина

Розглянемо класичну постановку завдання розпізнавання, т. е. віднесення деякого об'єкта одного з відомих класів.

Будемо вважати, що образ заданий точкою r_e-вимірного евклідового простору R^n, тобто є n-вимірним вектором, компоненти якого є дійсними числами. При цьому безліч точок, що відповідають одному класу образів, групується в деякій області вимірів простору. Кожна вісь такого простору співвідноситься з одним із re входів або з одним з re рецепторів системи, що розпізнає. Кожен рецептор може бути в одному з m станів, якщо вони дискретні, або мати нескінченно

велику кількість станів, якщо рецептори безперервні. Залежно від виду рецепторів розглядається дискретний, безперервний або змішаний (безперервно-дискретний) простір.

У просторі образів вводиться *метрика* - функція, яка кожній упорядкованій парі точок x і біля простору ставить у відповідність дійсне число. Метрика повинна мати такі властивості:

1. $d(x, y) \geq 0$, $(d(x, y)) = 0$ при $x = y$.
2. $d(x, y) = d(y, x)$.
3. $d(x, y) < d(x, z) + d(z, y)$.

Введення метрики дозволяє говорити про ступінь близькості точок простору, відповідно про міру подібності або відмінності образів.

Розглянемо, наприклад, завдання у просторі R^2. Нехай є чотири класи образів: А, Б, В та Г. Примірник кожного класу визначається точкою двовимірного простору, заданого координатами x_1 та x_2 (рис. 2.4).

Припустимо, що є два об'єкти: M_1 і M_2, які потрібно класифікувати. Очевидно, що класифікація у такій постановці зводиться до обчислення відстані (метрики) від об'єкта до кожного з наявних класів. Об'єкт належить до того класу, котрого ця відстань мінімально.

Відповідно до рис. 2.4 M_2класифікується однозначно, у той час як щодо приналежності M_1 виникає невизначеність, оскільки неможливо провести пряму, по один бік якої точки знаходилися б ближче до класу А, а по інший - ближче до класу Б (інакше кажучи, класи А та Б - *несепарабельні*). Для решти пар (А - Г, А - В, Б - Г, Б - В, Г - В) така пряма (її називають *роздільною* прямою) може бути проведена.

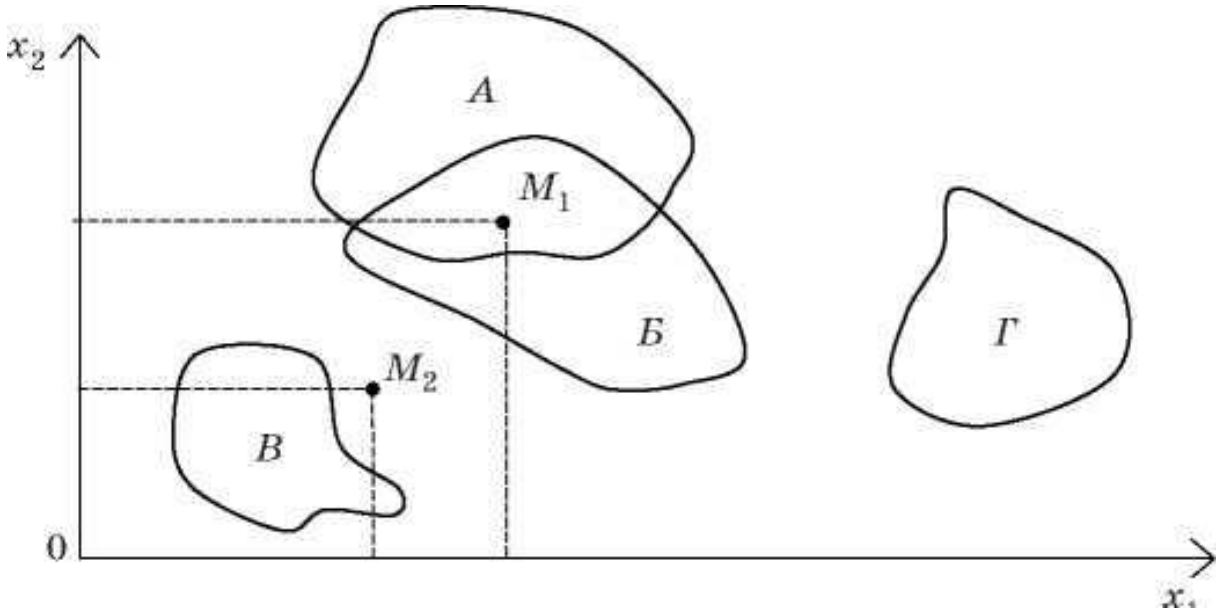


Рисунок 2.4 - Задача класифікації

Її рівняння має вигляд

$$s(x) = w_1 x_1 + w_2 x_2 + w_3 = 0 \qquad (2.1)$$

де w_1, w_2, w_3 - параметри, а x_1, $ч_2$- координати на плоскості.

Для точок, які лежать вище за цю лінію, $s(X) > 0$, а для тих, які лежать нижче за неї, $s(X) < 0$.

Розглянемо деякі два класи, наприклад, А та Б (див. рис. 2.5).

У разі гіперпростору розділяюча пряма перетворюється на гіперповерхню

$$s(x) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + w_{n-1} = WX + w_{n-1} = 0 \qquad (2.2)$$

Така гіперповерхня також поділяє гіперпростір на дві частини. Як буде показано далі, формули (2.1) та (2.2) є окремими випадками опису штучного нейрона. Вираз (2.2) іноді називають *лінійною машиною* (ЛМ).

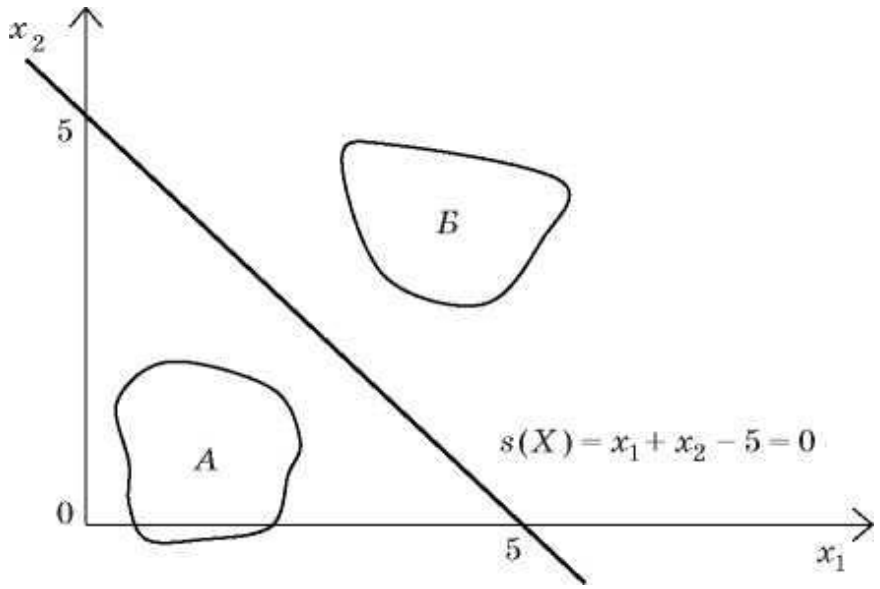


Рисунок 2.5 - Використання для класифікації розділяючої прямої

Розглянемо, яким чином ЛМ виконує класифікацію векторів мінімуму дистанції до центру кластера (під кластером тут розуміється група точок, які розташовані один до одного ближче, ніж до точок інших кластерів).

Нехай у P_i-вимірному евклідовому просторі відстань між двома точками вимірюється за формулою

$$\|X_i - X_j\| = \left[(X_i - X_j)^T (X_i - X_j)\right]^{\frac{1}{2}}$$

Позначимо через P_i центр тяжкості і кластера (i = 1, 2, ..., N). У процесі класифікації потрібно обчислити відстань від векторе вхідного X до центру кожного кластера і вибрати мінімальну відстань.

Розглянемо квадрат відстані до центру і кластера:

$$\|X - P_i\| = (X - P_i)^T (X - P_i) = X^T X - 2X^T P_i + P_i^T P_i, \qquad (2.3)$$

(так як $(A + B)^T = A^T + B^T$ та $X^T P = P^T X$).

Перший доданок (2.3) не залежить від i, отже, мінімізація (2.3) означає максимізацію зворотної функції:

$$d_i(X) = X^T P_i - \frac{1}{2} P_i^T P_i. \qquad (2.4)$$

Вираз (2.4) можна переписати як

$$d_i(X) = W_i^T X + w_{i,n+1}. \qquad (2.5)$$

де

$$w_{i,j} = p_{i,j}, \qquad w_{i,n+1} = -\frac{1}{2} P_i^T P_i$$

Розглянемо приклад. Нехай дано двовимірний простір із центрами кластерів, заданими векторами

$$P_1 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 6 \\ -5 \end{bmatrix}, \quad P_1 = \begin{bmatrix} -8 \\ 10 \end{bmatrix}$$

На підставі (2.5) можемо записати

$$d_1(X) = 3x_1 + 3x_2 - 9,$$
$$d_2(X) = 6x_1 - 5x_2 - 30.5,$$
$$d_3(X) = -8x_1 + 10x_2 - 82,$$

Структуру системи, що класифікує, ілюструє рис. 2.6.

Таким чином, до першого класу належать ті точки площини, для яких виконуються нерівності ($d_1(X) > d_2(X)$) та ($d_1(X) > d_3(X)$) або

$$\begin{cases} -3x_1 + 8x_2 - 21{,}5 > 0, \\ 11x_1 - 7x_2 + 73 > 0. \end{cases}$$

Аналогічно можна сформулювати умови приналежності до другого та третього класів.

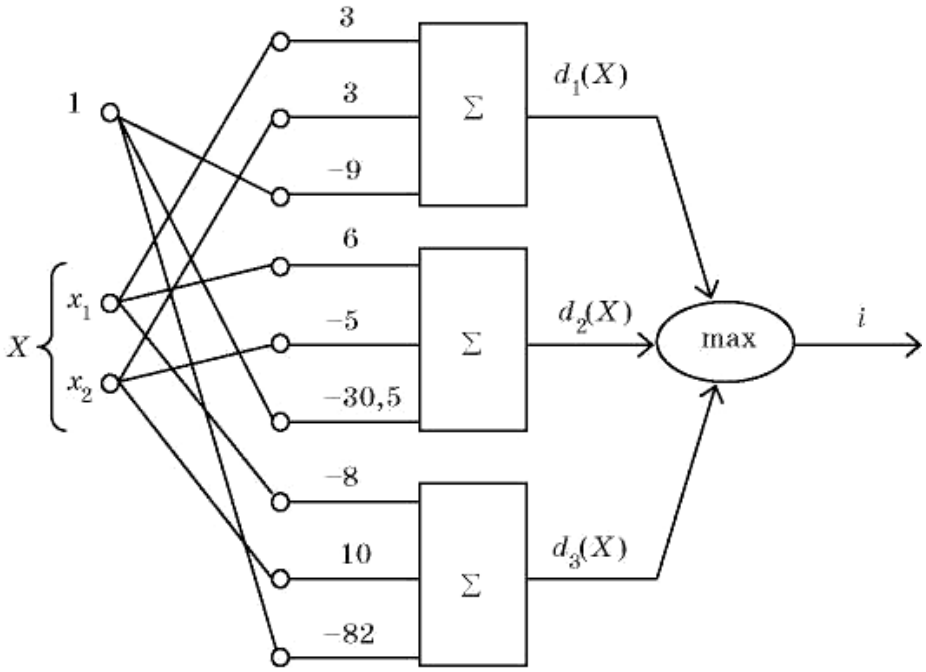


Рисунок 2.6 - Приклад використання для класифікації лінійної машини

2.5 Штучний нейрон

Штучний нейрон (ШН) - це найпростіший аналоговий елемент, що перетворює, моделює базові уявлення про роботу живого нейрона.

На вхід ШН надходить кілька сигналів. Кожен вхід зважується – множиться на певний коефіцієнт (синаптичну силу). Сума всіх творів визначає рівень активації нейрона. Підсумовуючий блок відповідає сомі живого нейрона (див. рис. 2.7).

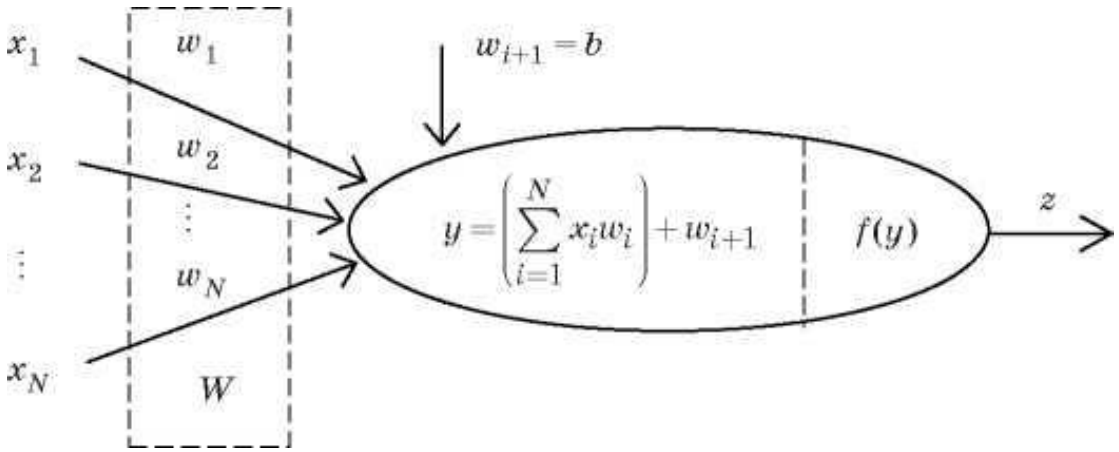


Рисунок 2.7 - Штучний нейрон

Активаційна функція *F* має бути монотонною. Зазвичай *F*(*y*) належить до інтервалу [0,1] або [-1,1]. Найчастіше використовують безліч варіантів активаційних функцій (табл. 1.2).

Таким чином, ШН виконує дві операції. Спочатку обчислюється сума скалярного добутку вектору ваг *W* та вхідного вектору *X*:

$$y = X^T W + b.$$

Потім спрацьовує активаційна функція, що визначає значення вихідного сигналу:

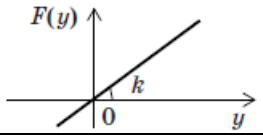
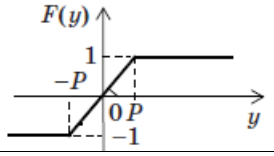
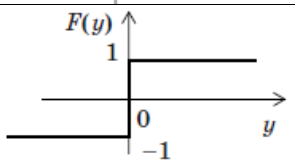
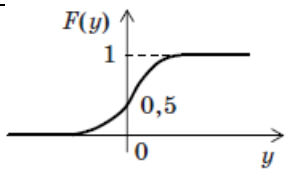
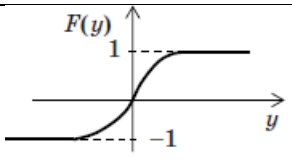
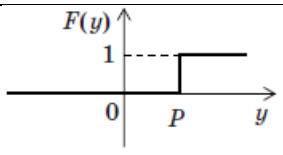
$$z = f(y).$$

Наприклад, якщо використовувати знакову активаційну функцію (див. табл. 2.2), вихід ШН можна описати формулою

$$f(y) = \mathrm{sgn}(X^T W + b).$$

Таким чином, ШН реалізує розділяючу пряму вигляду (2.2).

Таблиця 2.2 – Основні варіанти опису активаційної функції

Назва функції	*Опис*	*Графічне уявлення*
Лінійна	$F(y) = ky,$ $k > 0$ k – коефіцієнт активації	
Лінійна з насиченням	$F(y) = \begin{cases} +1, & y > P \\ ky, & \lvert y\rvert < 1 \\ -1, & y < -P \end{cases}$	
Визначення знаку	$F(y) = sgn(y),$	
Уніполярна сигмоїдальна (s-подібна)	$F(y) = \dfrac{1}{1 + \exp(-ky)}$ $k > 0$	
Біполярна сигмоїдальна (гіперболічний тангенс)	$F(y) = th(ky)$ $k > 0$	
Порогова	$F(y) = \begin{cases} 1, & y \geq P, \\ 0, & y < P. \end{cases}$	

Модель нейрона в MatLab

Математична модель нейрона представляється у такому вигляді:

$$a = f(wp + b),$$

де *a* – вихідний сигнал, *p* – вхідний сигнал, *w* – вага входу, *b* – усунення аргументу функції нейрона, *f*(*) – функція активації нейрона.

Математичної моделі нейрона зіставлено структурну схему, наведену рисунку 2.8.

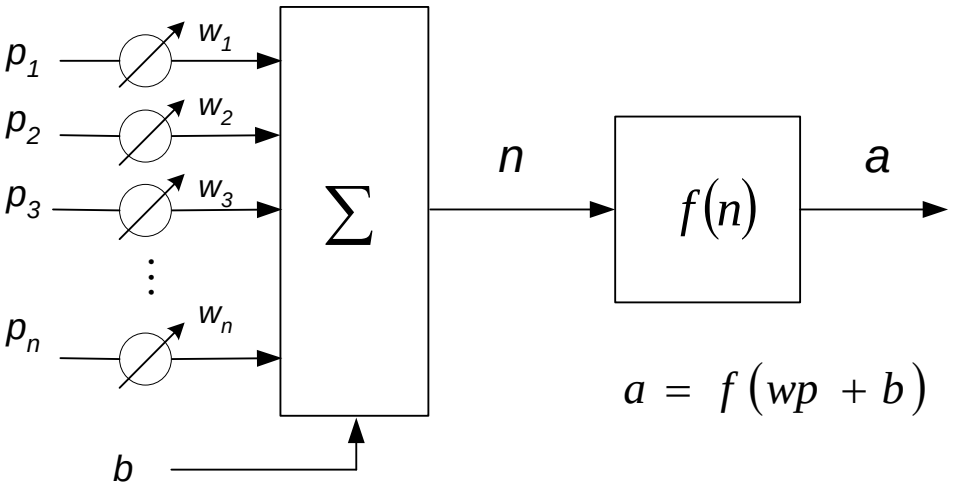


Рисунок 2.8 – Структурна схема моделі нейрона

Блок функціонального перетворення *f*(*n*) реалізує одну з функцій активації, приклади яких наведено на таблиці 2.2. Вигляд функції активації вибирається з урахуванням особливостей розв'язуваних завдань.

Структуру нейрона, показану рисунку 2.8, в Matlab прийнято зображати укрупненою схемою (див. рис. 2.9).

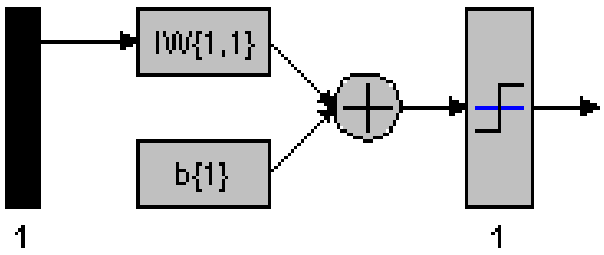


Рисунок 2.9 - Укрупнена структурна схема нейрона

Тут вхід нейрона зображується у вигляді темної вертикальної риси, під якою вказується кількість елементів входу R. Вектор входу множиться на вектор-рядок W довжини R. Зміщення b є скалярною величиною з константою 1. Результат підсумовування перетворюється функцією активації, умовно зображеної на вихідному блоці .

Паралельне включення нейронів утворює шар нейронної мережі.

2.6 Проблема лінійної роздільності

Розглянемо ШН з двома входами та пороговою активаційною функцією (див. рис. 2.10).

Усі можливі комбінації $x_1 w_1 + x_2 w_2$ визначають площину (x_1 x_2), а рівняння (2.1) – деяку пряму у цій площині.

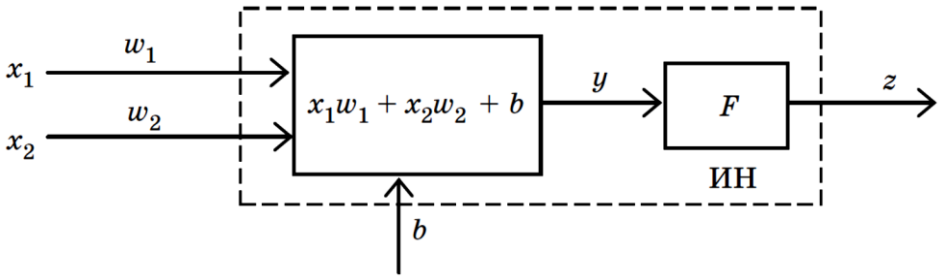


Рисунок 2.10 - ШН с двома входами

Розглянемо приклад. Нехай необхідно реалізувати логічні функції, задані у табл. 2.3.

Положення точок A_1 - A_4 на площині виглядає так (див. рис. 2.11).

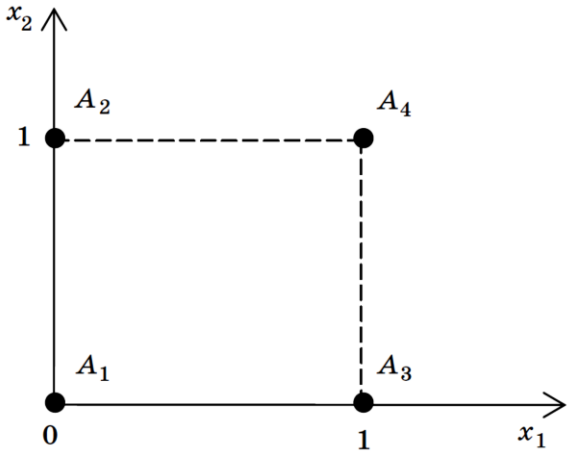


Рисунок 2.11 - Вхідні комбінації логічних функцій

Вочевидь, що з реалізації функції OR слід відокремити точку A_1 від інших точок, а реалізації функції AND необхідно відокремити точку A_4.

Таблиця 2.3 - Базові логічні функції

Точка	x_1	x_2	OR(x_1 x_2)	AND(x_1 x_2)	XOR(x_1 x_2)
A_1	0	0	0	0	0
A_2	0	1	1	0	1
A_3	1	0	1	0	1
A_4	1	1	1	1	0

Нехай входи x_1 та x_2 мають одиничні ваги: $w_1 = w_2 = 1$, а як F виберемо граничну функцію з порогом $P = 0$. Тоді для реалізації конкретної логічної функції слід вибрати зміщення b. Наприклад, за b = -1,5 виходить логічна функція AND (див. рис. 2.12).

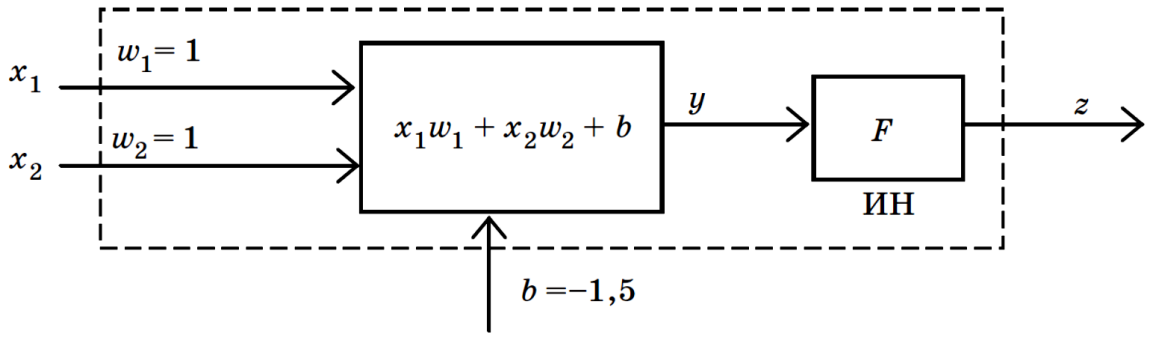


Рисунок 2.12 - Вхідні комбінації логічних функцій

При виборі b = -0,5 отримуємо логічну функцію OR.

На рис. 2.13 показані відповідні прямі розділяючі на площині.

Інша ситуація виникає під час спроби реалізувати функцію XOR.

У цьому випадку необхідно провести пряму, що розділяє таким чином, щоб точки A2 і A3 лежали в одній напівплощині, а точки A1 і A4 - в іншій. Це очевидно неможливо, тобто функція XOR є нереалізованою для одиничного нейрона з двома входами.

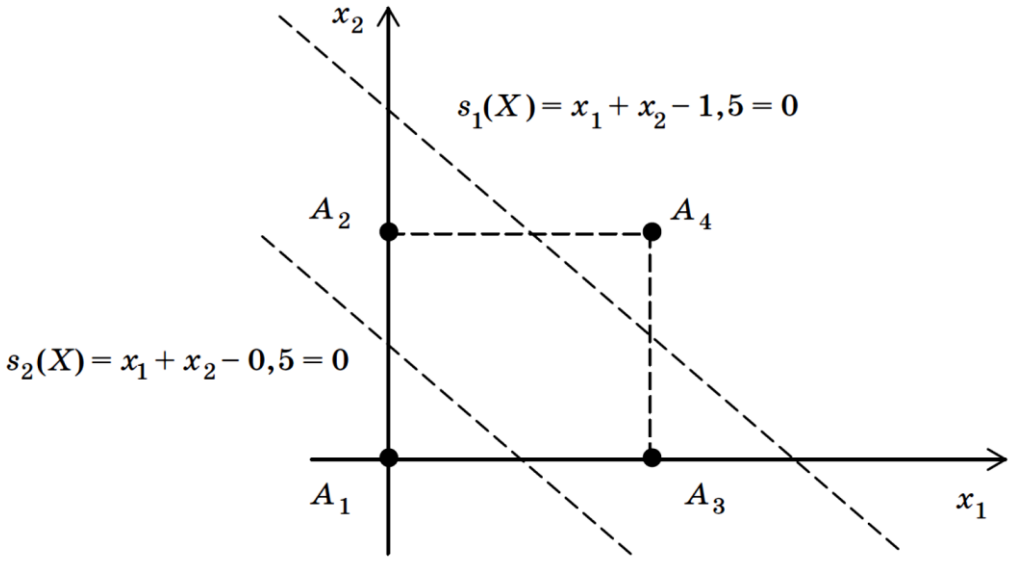


Рисунок 2.13 - Графічне представлення роботи нейронів AND та OR

Однак умова лінійної роздільності легко обійти, якщо ускладнювати нейронну структуру. Проблема функції XOR легко вирішується, якщо додати другий нейрон (див. рис. 2.14).

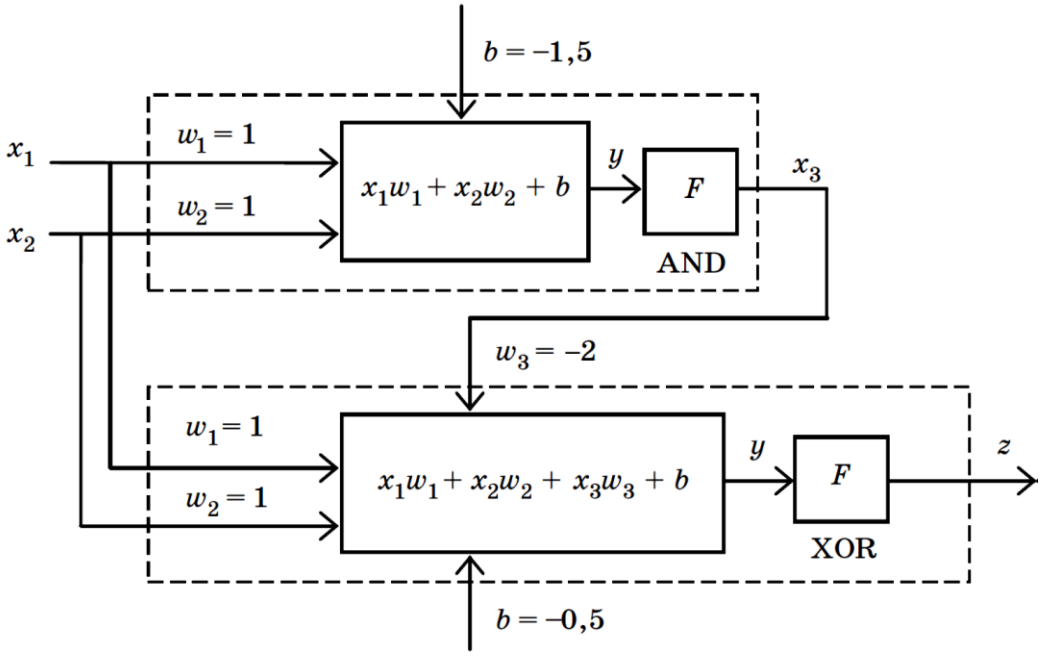


Рисунок 2.14 - Реалізація функції XOR за допомогою двох нейронів

Очевидно, що нейрон AND робить свій внесок тільки в ситуації $x_1 = x_2 = 1$, при цьому вихід нейрона XOR

$$z = F(1 + 1 – 0{,}5 – 2) = 0.$$

Реалізація функції XOR пов'язана з виділенням на площині області, що виходить в результаті комбінації двох прямих, що розділяють, тому структуру на рис. 2.14 правильніше зобразити у вигляді трьох нейронів (див. рис. 2.15).

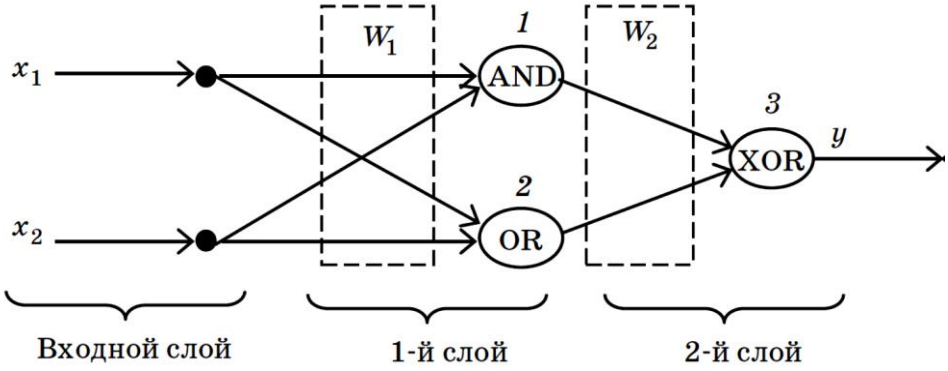


Рисунок 2.15 - ШНМ для опису функції XOR

Нейрон OR забезпечує роздільну пряму $s_1(X) = 0$, а нейрон AND - роздільну пряму $s_2(X) = 0$, як показано на рис. 2.16.

Для реалізації функції XOR нейрон 3 повинен видати одиничний сигнал лише при виконанні умови

($s_1(X) > 0$) AND ($s_2(X) < 0$),
тобто XOR(X,Y) = OR(X,Y) AND (NOT(AND(X,Y))).

Структура на рис. 2.15 являє собою найпростішу двошарову ШНМ.

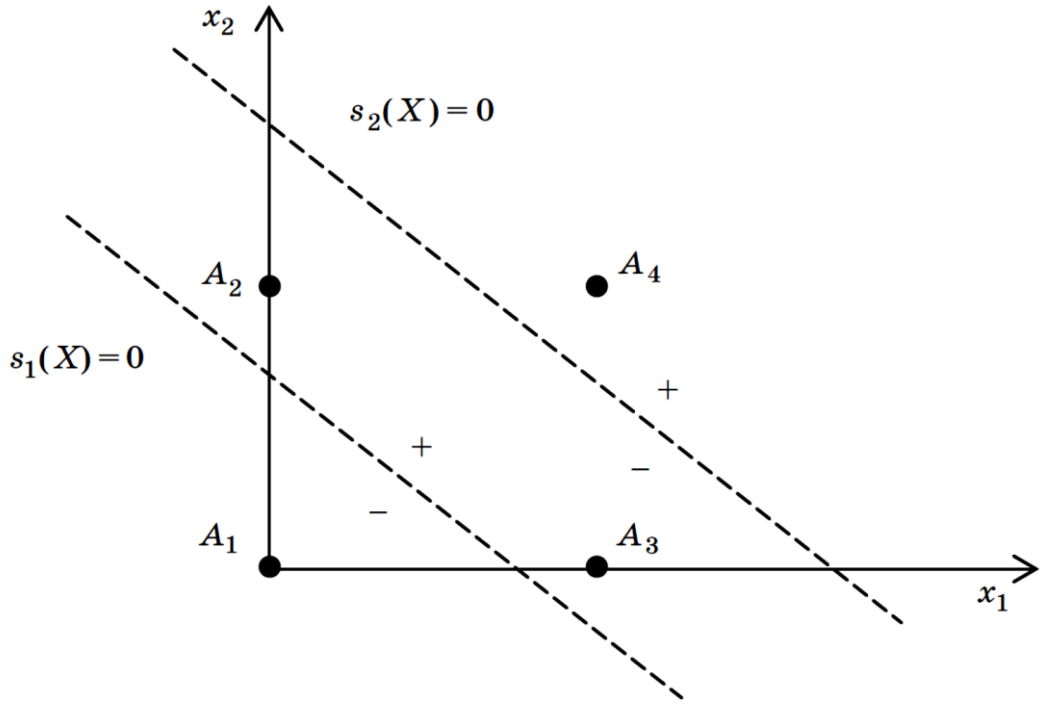


Рисунок 2.16 - Нейронна реалізація функції XOR

Рисунок 2.17 показує, що комбінуючи різні розділяючі прямі, на площині можна виділити область опуклої форми. Число розділяючих прямих відповідає числу нейронів 1-го шару.

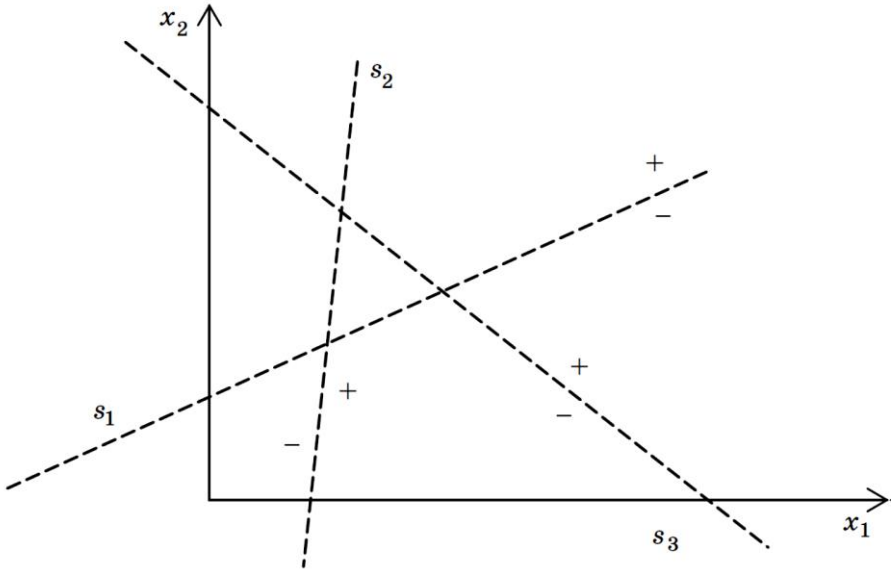


Рисунок 2.17 - Виділення опуклої області на площині

Тришарова мережа з двома входами дозволяє комбінувати різні опуклі фігури, отримуючи неопуклу фігуру (див. рис. 2.18).

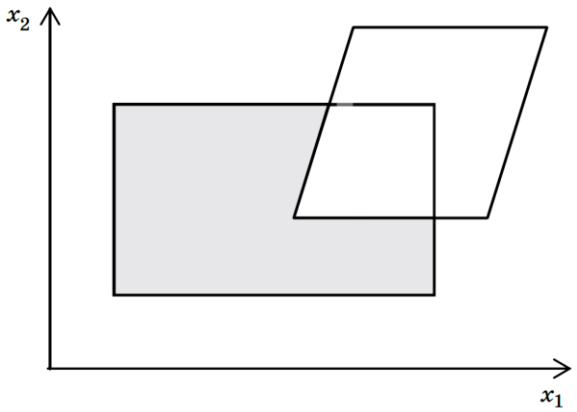


Рисунок 2.18 - Виділення на площині неопуклої області

При трьох входах ШНМ здатна виділяти задані області тривимірного простору, при чотирьох і більше входах йдеться про гіперпростір.

Зауважимо, що з двох логічних змінних можна описати 16 логічних функцій, у тому числі 14 є лінійно роздільними, т. е. можуть бути реалізовані з урахуванням ШН з двома входами. Для логічної функції трьох змінних частка лінійно роздільних функцій зменшується (104 з 256) і з подальшим зростанням кількості змінних різко зменшується.

Однак використання багатошарових ШНМ дозволяє вирішувати проблему лінійної роздільності. При цьому завдання вибору ваг і зміщень не може бути вирішена так само легко, як для функції XOR. В цьому випадку потрібна реалізація процедури навчання.

2.7 Правило навчання Хебба

Д. Хебб запропонував формальне правило, відповідно до якого вага w_{ij} зв'язку нейрона i з нейроном j змінюється пропорційно рівням їх збудження, тобто добутку їх вихідних сигналів:

$$\Delta w_{ij} = \eta y_i y_j$$

де $\eta \in [0,1]$ - коефіцієнт навчання.

Під час навчання з учителем замість вихідного сигналу yj використовується заданий вихідний сигнал z_j.

У кожному циклі навчання відбувається підсумовування поточного значення ваги та її збільшення Δw_{ij}:

$$\Delta w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

В результаті застосування правила Хебба ваги нейрона можуть набувати довільно великих значень. Один із способів стабілізації навчання за правилом Хебба полягає в обліку останнього значення w_{ij}, зменшеного на коефіцієнт забування γ. У цьому правило Хебба представляється як

$$\Delta w_{ij}(t+1) = w_{ij}(t)(1-\gamma) + \Delta w_{ij}$$

Значення γ найчастіше становить певний відсоток коефіцієнта навчання η.

Модифікацією правила навчання Хебба є дельта-правило, що базується на ідеї безперервної зміни синаптичних ваг для зменшення різниці (дельта) між значенням бажаного та поточного вихідного сигналу:

$$\Delta w_{ij} = \eta(z_j - y_j).$$

2.8 Концепція вхідної и виховної зірки

Поняття вхідної та вихідної зірки було введено С. Гроссбергом для аналізу процесу навчання нейронів.

Вхідна зірка (instar) складається з нейрона, на який подається група входів, помножених на синаптичні ваги (рис. 2.19).

Вхідна зірка повинна виконувати завдання розпізнавання, тобто її реакція повинна бути тим сильнішою, чим більше вхідний вектор *X* схожий на запам'ятований образ *W*. Якщо вхідний вектор і вектор ваги нормалізовані (мають одиничну довжину), то максимальне значення скалярного твору у виробляється при умови *X* = *W*.

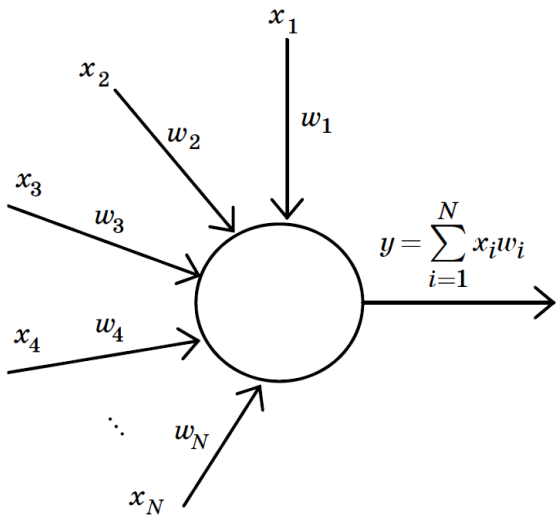


Рисунок 2.19 - . Вхідна зірка Гроссберга

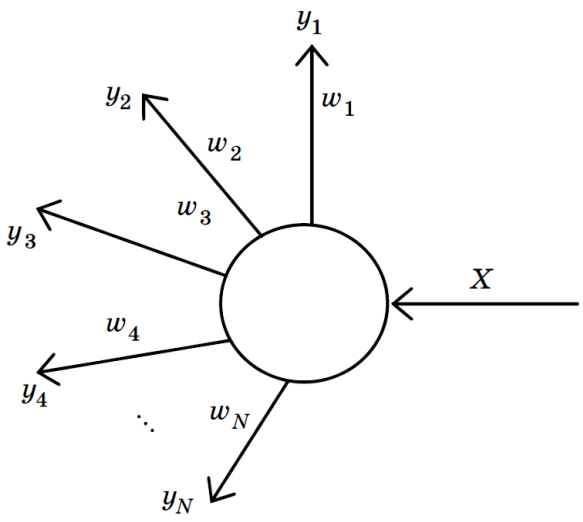


Рисунок 2.20 – Вихідна зірка Гроссберга

Для того щоб вхідна зірка реагувала не тільки на сам образ *X*, але і на його спотворені і зашумлені варіанти, формується безліч векторів, що незначно відрізняються від еталона.

Для кожного вектору з навчального набору корекція ваг

$$w_i^{t+1} = w_i^t + \alpha_t(x_i - w_i^t),$$

де t - момент часу; α_t - коефіцієнт швидкості навчання, що має початкове значення порядку 0,1 і поступово зменшується.

Таким чином, ваги вхідної зірки поступово осереднюються за набором навчальних векторів, наближаючись до центру кластера, який вони утворюють.

Вихідна зірка Гроссберга (outstar) є нейрон, керуючий групою ваг (рис. 2.20).

Вихідна зірка має видавати необхідний вектор *Y*.

Процес навчання вихідної зірки нагадує процес навчання вхідний. Він передбачає формування навчальної множини вихідних векторів з подальшою корекцією ваг:

$$w_i^{t+1} = w_i^t + \alpha_t(y_i - w_i^t),$$

Вхідні та вихідні зірки можуть з'єднуватися один з одним у різних конфігураціях.

2.9 Парадигми навчання

Під час навчання нейронної мережі (неявному побудові алгоритму розв'язання задачі) топологія мережі зазвичай вважається незмінною, а налаштуванню підлягають ваги зв'язків між нейронами.

За способом використання вчителя виділяють три основні методи навчання.

Навчання з учителем (supervised learning). У такому разі НМ надаються вхідні дані та бажані вихідні та вигляді набору навчальних пар. Оскільки для кожного вхідного вектора заданий вихідний вектор, НМ може коригувати ваги, порівнюючи поточні та еталонні виходи. У процесі навчання мають бути мінімізовані помилки з усіх пар. Зрозуміло, навчальна вибірка містить лише частину можливих варіантів вхідних сигналів, а мета навчання полягає в отриманні правильної реакції на будь-який вхідний допустимий сигнал. Тому дані навчальної вибірки лише задають опорні точки у просторі входів-виходів мережі, і мета навчання полягає у виконанні правильної інтерполяції (екстраполяції) між цими опорними точками.

Таким чином, правильне завдання навчальної вибірки також впливає на успіх процесу навчання.

Навчання без вчителя (unsupervised learning). У цьому випадку НМ сама аналізує особливості вхідних даних з метою пошуку структур та закономірностей. Ваги підлаштовуються так, щоб утворювалися

узгоджені вихідні вектори, тобто щоб при отриманні близьких входів виходили близькі виходи, і навпаки. У процесі навчання ШНМ виконує кластеризацію (групування) вхідної інформації, а після навчання мережа працює у режимі векторного класифікатора. Такий підхід характерний для НР Кохонена. Процедура навчання без учителя багато в чому відповідає процесам, що протікають у живій природі. Живі істоти у розвитку збирають подібні подразники групи, і з кожною групою пов'язується позитивна чи негативна реакція, т. е. навчання можна як узагальнення і стиск інформації.

Навчання з підкріпленням (reinforcement learning). Такий підхід займає проміжне положення між двома попередніми. У цьому випадку замість еталонного значення виходу НМ використовується оцінка, що формується зовнішнім середовищем. Ця оцінка має невисоку інформативність, тому можна сказати, що система, що навчається, отримує набагато менше інформації, ніж при навчанні з вчителем, і трохи більше, ніж при навчанні без вчителя. Навчання з підкріпленням є складним процесом, заснованим на методі спроб та помилок. Основною сферою застосування цієї парадигми є завдання, у яких необхідно знайти оптимальну послідовність дій. Такі завдання характерні при нейромережевому управлінні автономними агентами, наприклад, робототехніки.

По суті, завдання навчання ШНМ є завданням оптимізації, тому можлива ще одна класифікація, яка ділить всі методи навчання ШНМ на дві групи:

- Детерміновані методи навчання. Цими методами процедура корекції ваг НМ здійснюється крок за кроком на підставі інформації про вхід мережі, її поточний та бажаний вихід. Такі методи навчання є локальними, тобто припускають, що помилкова функція унімодальна;

- Стохастичні методи навчання. Такий підхід відповідає глобальній оптимізації, коли передбачається, що функція помилки може мати безліч екстремумів (мультимодальна функція). Для того щоб потрапити в околицю глобального екстремуму, потрібно здійснювати рухи, що тимчасово погіршують цільову функцію. З цією метою до формули корекції ваг вводиться випадковий фактор.

Таким чином, існують дві підгрупи методів навчання з учителем:

- До першої підгрупи належать методи поградієнтного спуску: метод найменших квадратів та метод зворотного поширення помилки. Обидва ці підходи локальні, т. е. адекватні при унімодальному характері функції помилки.

- до другої підгрупи належать стохастичні методи, такі, як генетичний алгоритм, роєвий алгоритм, метод відпалу металу і т. д. Ці методи працюють і за мультимодального характеру функції помилки, тобто є глобальними.

2.10 Попередня обробка інформації та оцінка якості роботи нейромережі

На вхід ШНМ надходить числова інформація, яка може мати різний зміст. За допомогою кодування можна описувати текстову інформацію, зображення та будь-які параметри досліджуваних об'єктів для ШНМ.

Якісні змінні (ознаки) набувають кінцевої кількості значень.

Якщо значень лише два, то така змінна кодується одним бітом. Якщо значень n, то знадобиться N біт:

$$N = \log_2 n.$$

Для опису якісних ознак може бути використана теорія нечітких множин, відповідно до якої для одного стану об'єкта можуть одночасно справедливі два або більше лінгвістичних значень, що мають різний ступінь приналежності.

Успіх навчання ШНМ багато в чому залежить від підготовки кількісних навчальних даних, що зазвичай містить виняток аномальних спостережень, заповнення пропущених даних, а також нормування вхідних змінних.

Так, у попередньому прикладі розглядалася схема, в якій похідна береться від ступінчастої функції. У точці стрибка ця похідна набуває нескінченного значення, що може зробити навчання неможливим. Для виправлення ситуації достатньо замінити аномальне "нескінченне" значення "велике".

Відновлення пропущених даних може бути виконане за допомогою інтерполяції (екстраполяції), моделювання чи експертних оцінок.

Нормування означає приведення кожної компоненти вхідного вектора до інтервалу [0,1] або [-1,1]. При відомому діапазоні зміни вхідної змінної [x_{min}, x_{max}] нормування виконується за формулою

$$x_N = \frac{x - x_{min}}{x_{max} - x_{min}}.$$

Якщо значення вхідної змінної потрібно привести до заданого інтервалу [a, b], то застосовується формула

$$x_N = \frac{(x - x_{min})(b - a)}{x_{max} - x_{min}} + a.$$

Іноді значення x_{max} є дуже великим: $x_{max} \to \infty$. У такій ситуації можна використати формулу

$$x_N = \frac{1}{1 + e^{-x}}.$$

Якщо потрібно враховувати малі значення великих величин, то можна використовувати так звану *модулярну обробку даних*, коли число *x* замінюється вектором *Z*, кожну компоненту якого знаходять за формулою

$$z_i = \frac{\big((x mod y_i) + y_i\big)(b-a)}{2y_i} + a.$$

де z_i - *i*-я компонента векторе *Z*; y_i - позитивні цілі числа із заданого набору.

Залежно від особливості розв'язуваного завдання корисними можуть виявитися різноманітні перетворення як окремих змінних (зведення у ступінь, вилучення кореня, взяття зворотних величин та інш.), і їх комбінацій (суми, добуток, приватні похідні).

При оцінці якості роботи ШНМ зазвичай потрібно розглядати середню квадратичну помилку, що визначається як усереднена на прикладах n сума квадратів різниць між бажаною величиною виходу z_i та реально отриманими на мережі значеннями y_i для кожного *i*-го прикладу:

$$E = \frac{1}{N}\sum_{i=1}^{N}(z_i - y_i)^2.$$

У завдання класифікації різні групи прикладів можуть мати різну вагу або самі приклади можуть мати різну значимість. Тоді до останньої формули доцільно ввести вагові коефіцієнти:

$$E = \frac{1}{N}\sum_{i=1}^{N} k_i(z_i - y_i)^2.$$

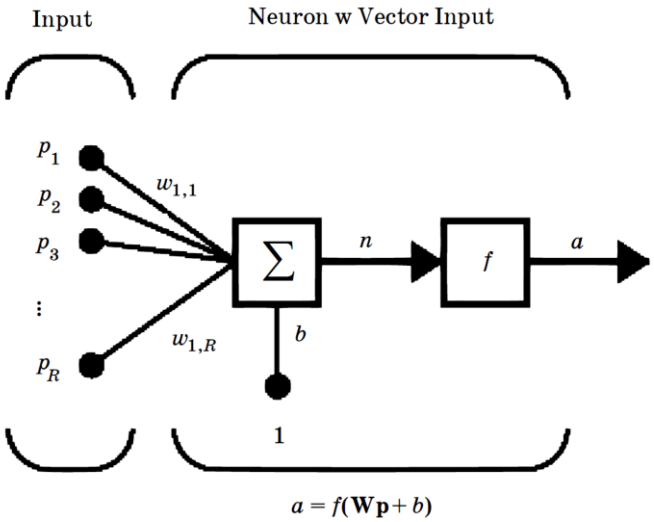
При апаратній реалізації ШНМ часто застосовують спрощену оцінку:

$$E = \sum_{i=1}^{N}|z_i - y_i|\text{, або } E = max_i|z_i - y_i|.$$

3 ОДНОШАРОВІ НЕЙРОННІ МЕРЕЖІ

3.1. Опис штучного нейрона у MatLab

На рисунку 3.1 представлений штучний нейрон у системі MatLab.



p - вектор-стовпець входу, W - матриця ваги (для одного нейрона це вектор-рядок), f - активаційна функція, a - вихід нейрона

Рисунок 3.1 - Структура нейрона в MatLab

При описі роботи з пакетом Neural Net Toolbox MatLab використовується ряд загальних позначень. Так, скаляри позначаються курсивними малими літерами (*a*, *b*, *c*, ...), вектори - прямими малими напівжирними літерами (**a**, **b**, **c**, ...), матриці - прямими великими напівжирними літерами (**A**, **B**, **C**, ...).

Найпростіша ШНМ з одного нейрона з лінійною активаційною функцією може бути створена командою

```
>> net = newlin([0 10; 0 10],1);
```

Тут матриця [0 10; 0 10] описує діапазони вхідних значень для кожного входу ШНМ (тобто число рядків відповідає числу входів), цифра вказує число нейронів; net - довільна назва створюваної мережі.

Ваги та усунення нейрона можуть бути задані командами:

```
>> net.IW{1,1}=[3 4]
>> net.b{1}=1
```

Тут IW - матриця ваги вхідного шару (він же є першим шаром). Вираз у дужках {n, m} вказує на те, що ваги відповідають зв'язку від шару m до шару n. Вираз b{n} означає усунення *n*-го шару.

Для моделювання роботи мережі слід описати деякий вхідний вектор-стовпець, довжина якого дорівнює числу входів:

```
>> P=[1; 3]
P =
1
3
```

Потім виконується моделювання роботи мережі:

```
>> X=sim(net,P)
X =
16
```

Можна описати послідовність вхідних векторів:

```
>> P=[[1; 3],[2;4],[3;8]]
P =
1      2      3
3      4      8
```

Тоді вихід мережі буде вектором

```
>> X=sim(net,P)
X =
16 23 42
```

Варіанти активаційних функцій наведено у табл. 3.1.

Таблиця 3.1 - Варіанти опису активаційної функції у MatLab

Функція активації	Опис функції
hardlim(x)	Порогова з порогом 0
hardlims(x)	Біполярна порогова з порогом 0
purelin(x)	Лінійна
logsig(x)	Логістична (сигмоїдна)
poslin(x)	Позитивна лінійна
satlin(x)	Позитивна лінійна з насиченням
satlins(x)	Біполярна лінійна з насиченням
radbas(x)	Радіальна базисна
tribas(x)	Трикутна базисна
tansig(x)	Гіперболічний тангенс

3.2 Персептрон

Персептрон (від латів. perceptio – сприйняття) – найпростіша штучна нейронна мережа, виявлена в результаті багаторічних досліджень мозку людини та тварин [3]. Кожен із п'яти видів інформації про довкілля (зорова, слухова, дотикова, нюхова, смакова)

сприймається своїми спеціалізованими сенсорними нейронами і передається окремими сенсорним трактам в центральну нервову систему. Розглянемо модель зорової системи (рис. 2.2).

Модель включає три послідовно з'єднаних безлічі нейронів: чутливі *S*-елементи, асоціюючи *A*-елементи і реагують *R*-елементи.

Чутливим елементам відповідають сенсорні нейрони, що асоціюють - локальні спеціалізовані зорові центри в корі мозку і реагують - моторні нейрони, що передають керуючі сигнали м'язам та залозам організму. Можна вважати, що Д-елемент реагує на певний клас зображень.

Сенсорні нейрони збуджуються від впливу енергії світла за перевищення деякого порога. У свою чергу *A*-елементи порушуються лише тоді, коли порушено достатню кількість *S*-елементів, пов'язаних із ними. *R*-елемент збуджується, якщо з його вході виникає певна комбінація *A*-елементів.

Зауважимо, що під персептроном розуміється зазвичай одношарова нейронна мережа. Багатошаровий персептрон називається нейронною мережею прямого розповсюдження.

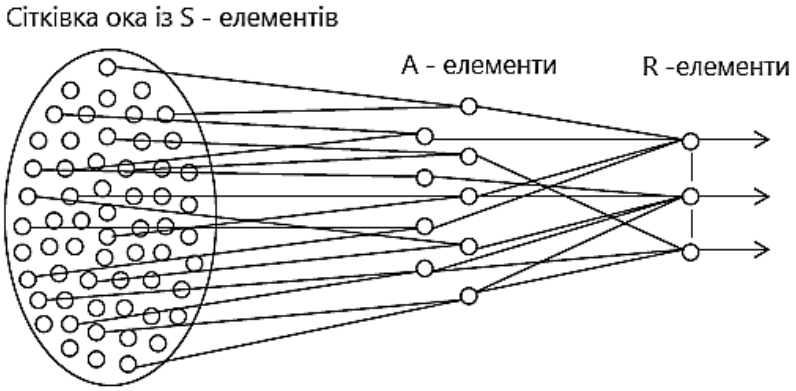


Рисунок 3.2 - Структура моделі зорової системи

Розглянемо персептрон з одним виходом, т. е. практично один нейрон, має безліч вхідних сигналів. Завдання навчання персептрона полягає у збудженні його лише за умови пред'явлення певних вхідних векторів та відсутності реакції на інші вхідні вектори.

Таким чином, під час навчання персептрона вирішується завдання бінарної класифікації - шляхом налаштування ваг персептрона будується гіперплощина, яка розділяє всі вхідні вектори на два класи.

Наприклад, нехай потрібно навчити персептрон розпізнавати парні та непарні цифри за їхніми зображеннями. Еталонне зображення кожної цифри розбивається на сегменти із заданим ступенем подробиці. Якщо межі сегмента потрапляє частина зображення, йому відповідає одиничний сигнал, інакше - сигнал нульовий. Так, кожна цифра одержує відповідний бінарний код. Приклад цифри «5» наведено

на рис. 3.3.

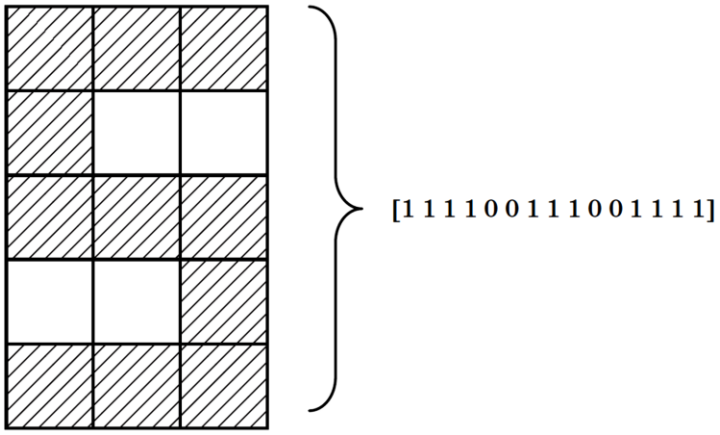


Рисунок 3.3 - Кодування зображення цифр

Таким чином, отримуємо десять навчальних пар, що описують вхід та бажаний вихід персептрону (див. табл. 3.2). Оскільки бажане рішення заздалегідь відоме, помилку роботи ШНМ можна вираховувати у явному вигляді, і йдеться про навчання з учителем.

Таблиця 2.2 – Навчальні пари

Цифра	Вхідний вектор X	Признак парності 2	Двійковий код цифри Z
0	[1 1 1 1 0 1 1 0 1 1 0 1 1 1 1]	1	0000
1	[0 1 0 0 1 0 0 1 0 0 1 0 0 1 0]	0	0001
2	[1 1 1 0 0 1 1 1 1 1 0 0 1 1 1]	1	0010
3	[1 1 1 0 0 1 1 1 1 0 0 1 1 1 1]	0	0011
4	[1 0 1 1 0 1 1 1 1 0 0 1 0 0 1]	1	0100
5	[1 1 1 1 0 0 1 1 1 0 0 1 1 1 1]	0	0101
6	[1 1 1 1 0 0 1 1 1 1 0 1 1 1 1]	1	0110
7	[1 1 1 0 0 1 0 1 0 1 0 0 1 0 0]	0	0111
8	[1 1 1 1 0 1 1 1 1 1 0 1 1 1 1]	1	1000
9	[1 1 1 1 0 1 1 1 1 0 0 1 1 1 1]	0	1001

Сенс використання персептрона полягає в тому, що після навчання на його вхід можуть надходити вже не еталонні, а спотворені та зашумлені вектори. Реакція ШНМ має бути правильною до досягнення певного максимального рівня шумів. Як випливає з табл. 2.2 деякі образи розрізняються тільки значенням одного біта, так що завдання розпізнавання не вдасться вирішити вже при мінімальному шумі, тому в реальності опис повинен бути більш докладним.

При навчанні персептрон отримує по черзі вхідні вектори відповідно до рисунку 3.4.

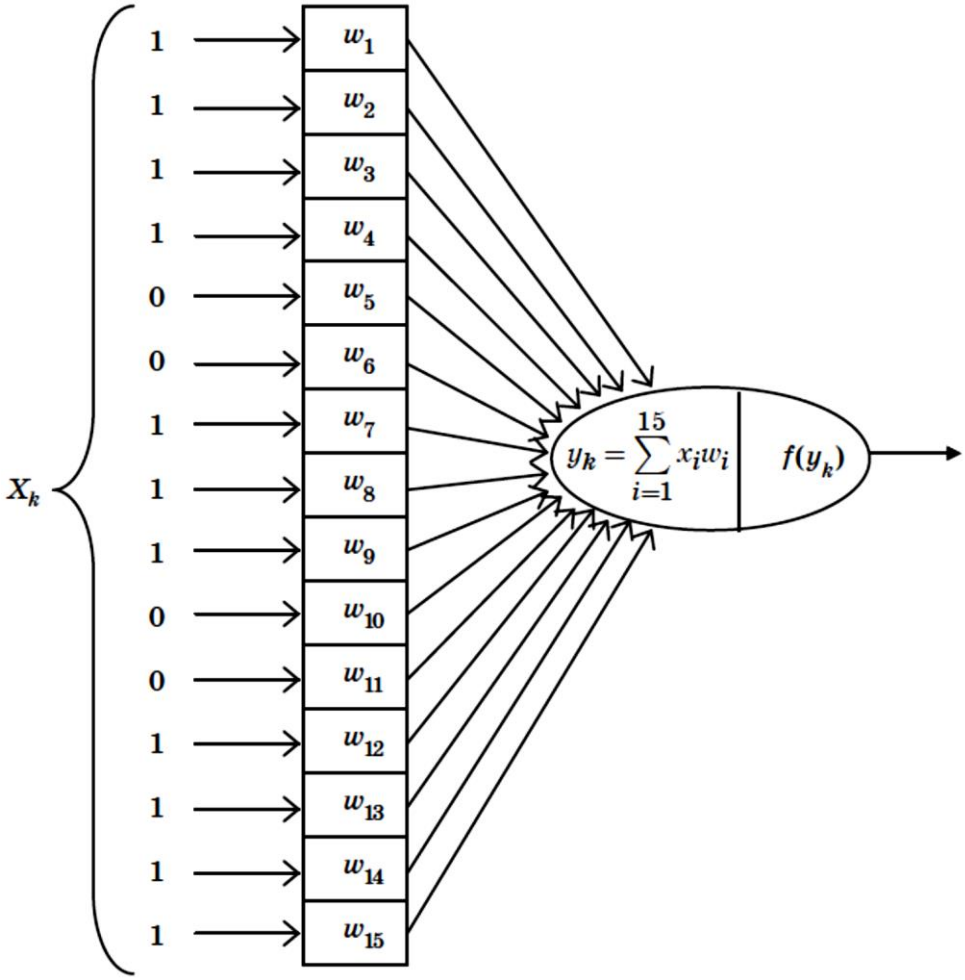


Рисунок 3.4 - Навчання персептрона

Помилка навчання в розглянутому завданні можна описати формулою

$$E = \sum_{k=0}^{9} |e_k| = \sum_{k=0}^{9} |f(y_k) - z_k|, \qquad (3.1)$$

де $f(y_k)$ та z_k - реальний та заданий вихід персептрона для k-го навчального входу.

Активаційна функція вибирається граничною. Спочатку матриця вагових коефіцієнтів **W** отримує випадкові значення.

Алгоритм навчання персептрону зводиться до таких дій:

1. На вхід подається *k*-й образ навчального набору.
2. Обчислюється помилка $e_k = f(y_k) - z_k$.
3. Якщо $e_k = 0$, то вагові коефіцієнти **W** не змінюються.
4. Якщо $e_k \neq 0$, то змінюються ті вагові коефіцієнти, які посилюють помилку (w_i, яким відповідають ненульові значення x_i):

$$w_i \leftarrow (w_i - sgn(e_k)x_i)$$

(при такому описі крок навчання дорівнює одиниці, і поріг нейрона має бути в кілька разів більшим).

5. Помилка e_k запам'ятовується. Кроки 1-4 повторюються по всіх повчальних парах. Обчислюється помилка навчання (3.1). Якщо $E > 0$, відбувається перехід крок 1.

Якщо завдання відноситься до класу реалізованих на персептроні, то описаний алгоритм дозволить налаштувати вагові коефіцієнти бажаним чином.

Послідовність подачі на вхід ШНС всіх навчальних прикладів називається *епохою*.

Персептрон може мати більше одного виходу. Наприклад, якщо ми хочемо розпізнати конкретну цифру, то на виході персептрона має з'являтися двійковий код цієї цифри, що складається із чотирьох бітів (див. табл. 3.2). Відповідно буде потрібно чотири ШН (рис. 3.5).

Формула помилки навчання матиме вигляд

$$E = \sum_{k=0}^{9} \Delta_k = \sum_{k=0}^{9} \left(\sum_{j=1}^{4} \delta_{kj} \right) = \sum_{k=0}^{9} \left(\sum_{j=1}^{4} \left| f\left(y_{kj}\right) - z_{kj} \right| \right). \qquad (3.2)$$

Активаційна функція також вибирається пороговою, а матриця **W** отримує спочатку випадкові значення.

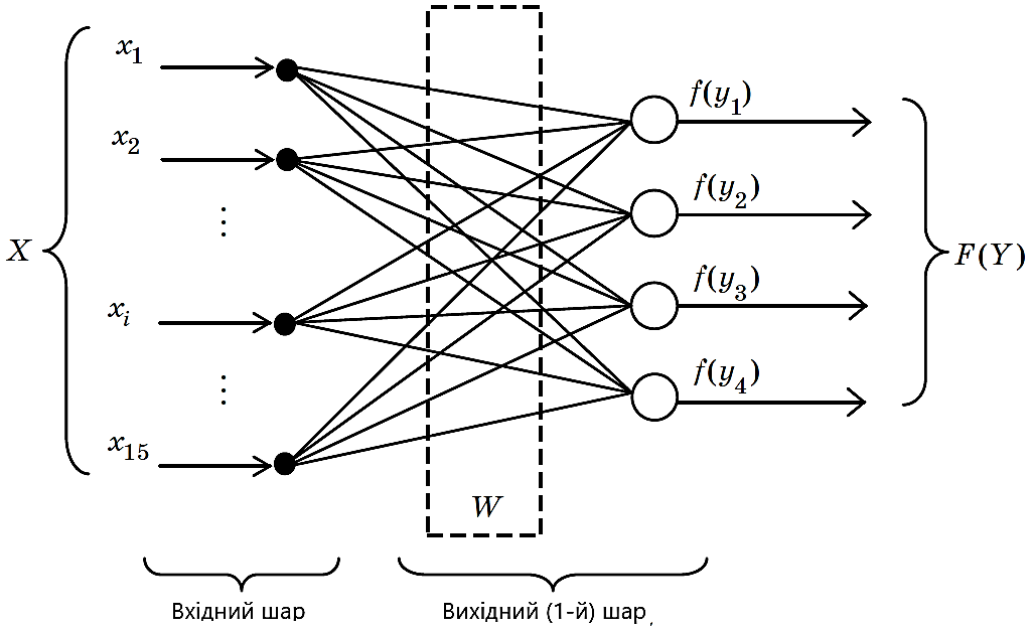


Рисунок 3.5 - Персептрон з безліччю вихідних нейронів

Алгоритм навчання персептрона дещо змінюється:

1. На вхід подається *k*-й образ навчального набору.
2. Обчислюється помилка першого нейрона:

$$\delta_{k1} = f(y_{k1}) - z_{k1}.$$

3. Коригуються вагові коефіцієнти нейрона, що посилюють помилку виходу.

4. Кроки 2 - 3 повторюються інших нейронів.

5. Δ_k запам'ятовується. Кроки 1-4 повторюються по всіх повчальних парах. Обчислюється помилка навчання (2.2). Якщо *E* > 0, відбувається перехід крок 1.

Зауважимо, що спосіб кодування зображення показаний на рис. 3.3, на практиці не застосовується, тому спотворення всього одного біта призводить до неправильного розпізнавання. У реальних завданнях розглядаються вектори більшої розмірності.

Входи та виходи персептрону можуть бути дійсними числами. У цьому випадку закон навчання персептрона узагальнюється у вигляді так званого *дельта-правила*, відповідно до якого алгоритм корекції ваги зв'язку між і входом і j виходом записується в наступному вигляді:

$$w_{ij}(k+1) = w_{ij}(k) + \eta\delta_{kj}x, \qquad (3.3)$$

де *n* - коефіцієнт швидкості навчання (зазвичай $\eta \in [0.01, 0.1]$), який поступово зменшується.

Вираз (3.3) показує, що величина корекції ваги пропорційна відповідній компоненті вхідного сигналу та помилки виходу.

На рисунку 3.6 наведено опис структури персептрону, що використовується в MatLab.

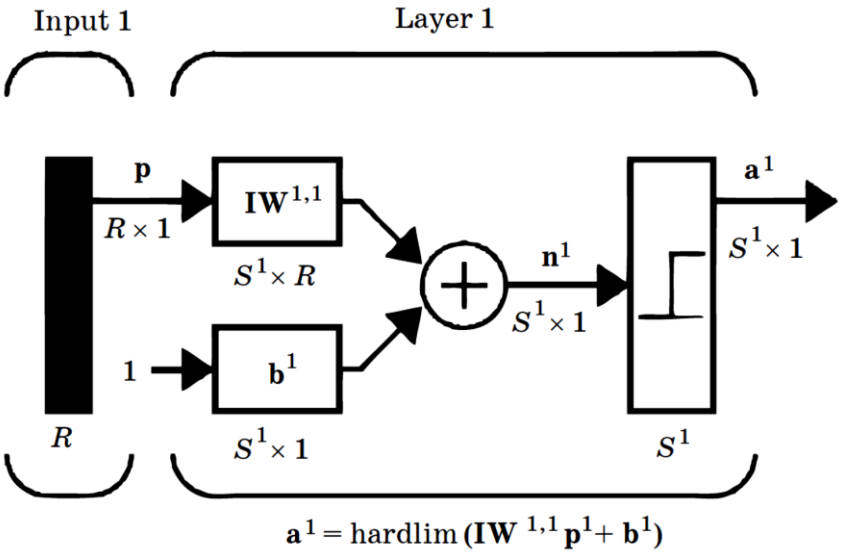


Рисунок 3.6 - Графічне зображення персептрону

Створення персептрону відбувається за командою

```
>> net = newp(PR, S)
```

де *PR* - матриця *R* ×2, що описує мінімальне та максимальне значення по кожному з *R* входів; *S* – число нейронів. Можна використовувати більш детальний опис:

```
>> net = newp(PR, S, tf, lf)
```

де tf - передаточна функція зі списку {hardlim, hardlims}, причому за замовчуванням задається hardlim; lf - навчальна функція зі списку {learnp, learnpn}, за замовчуванням - learnp.

При створенні персептрона матриця ваги і вектор зсувів ініціалізуються нулями за допомогою функцій initzero. Можна також проініціалізувати значення ваг та зміщень командами

```
>> net.IW{1,1} = [-1 1];
>> net.b{1} = [1]

>> untitled
net =
  struct with fields:
    IW: {[-1 1]}
     b: {[1]}
```

Навчання персептрона проводиться за допомогою функції адаптації adapt, яка коригує ваги та зміщення за результатами обробки кожної пари вхідних та вихідних значень (навчання з учителем):

```
>> adapt(net,P,T)
```

де *P* – вхідні вектори; *T* – цільові значення.

Застосування функції adapt гарантує, що будь-яка задача класифікації з лінійно відокремленими векторами буде вирішена за кінцеве число налаштувань.

Функція навчання train також може бути використана для навчання персептрона, але тут налаштування виконується не після кожного проходу, а в результаті всіх проходів навчальної множини. Кожен перерахунок набору вхідних векторів є епохою. Розглянемо приклад бінарної класифікації чотирьох векторів:

```
>>p = [[2;2] [1;-2] [-2;2] [-1;-0.5]];
>>t =[0 1 0 1];
```

За допомогою функції

```
>>plotpv(p,t);
```

вхідний та цільовий вектори можна відобразити графічно. Це допомагає зрозуміти, чи є завдання лінійно розділеним (див. рис. 3.7):

```
>>net = newp([-2 2;-2 2],1);
>>net.trainParam.epochs = 3;
>>net = train(net,p,t);
TRAINC, Epoch 0/100
TRAINC, Epoch 2/100
TRAINC, Performance goal met.
```

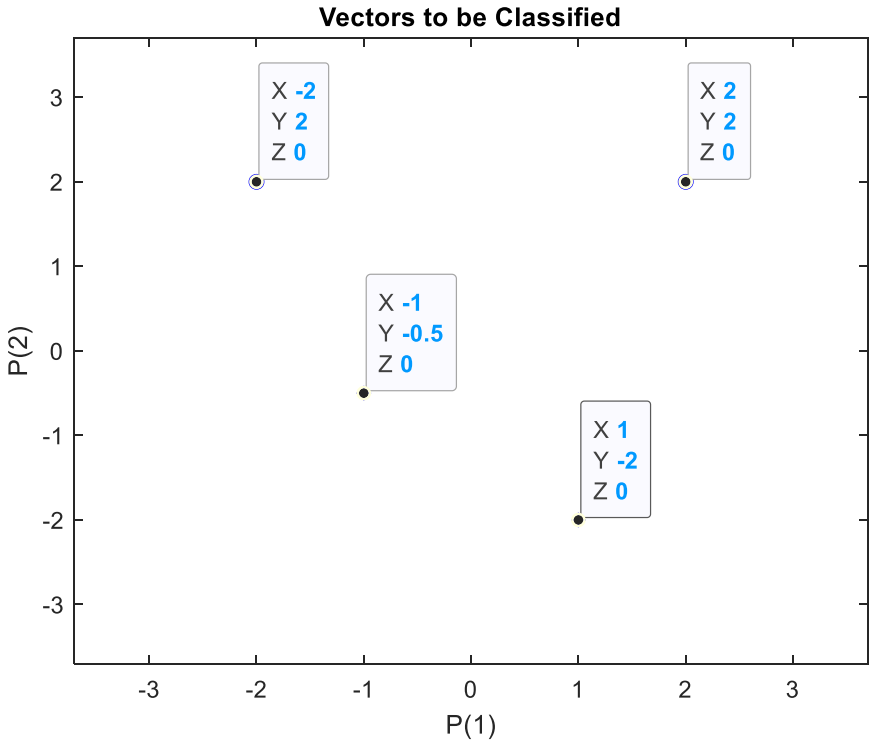
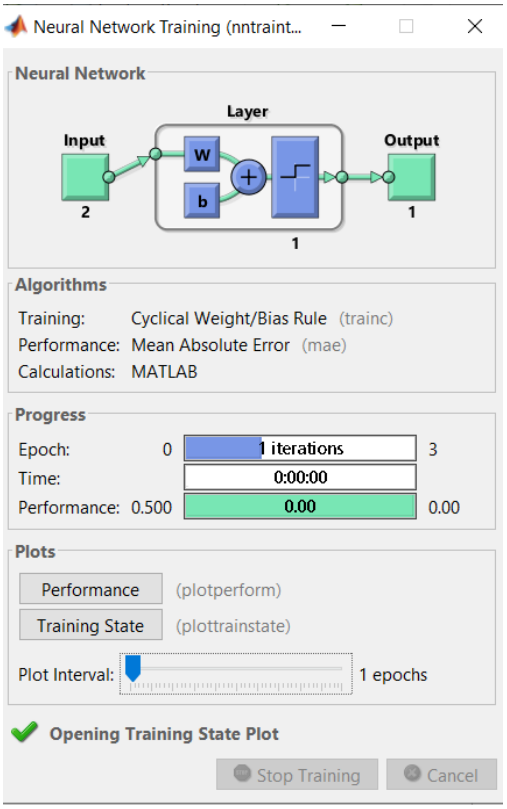


Рисунок 3.7 - Початкові дані для класифікації

Мнемонічне зображення з'являється на екрані, показує, що проблема була вирішена в три епохи (див. рис. 3.8).



Рисунко 3.8 - Графический протокол обучения персептрона

Використовуючи функцію plotpc, ви можете проаналізувати положення ділянки після тренувань (див. рис. 3.9)

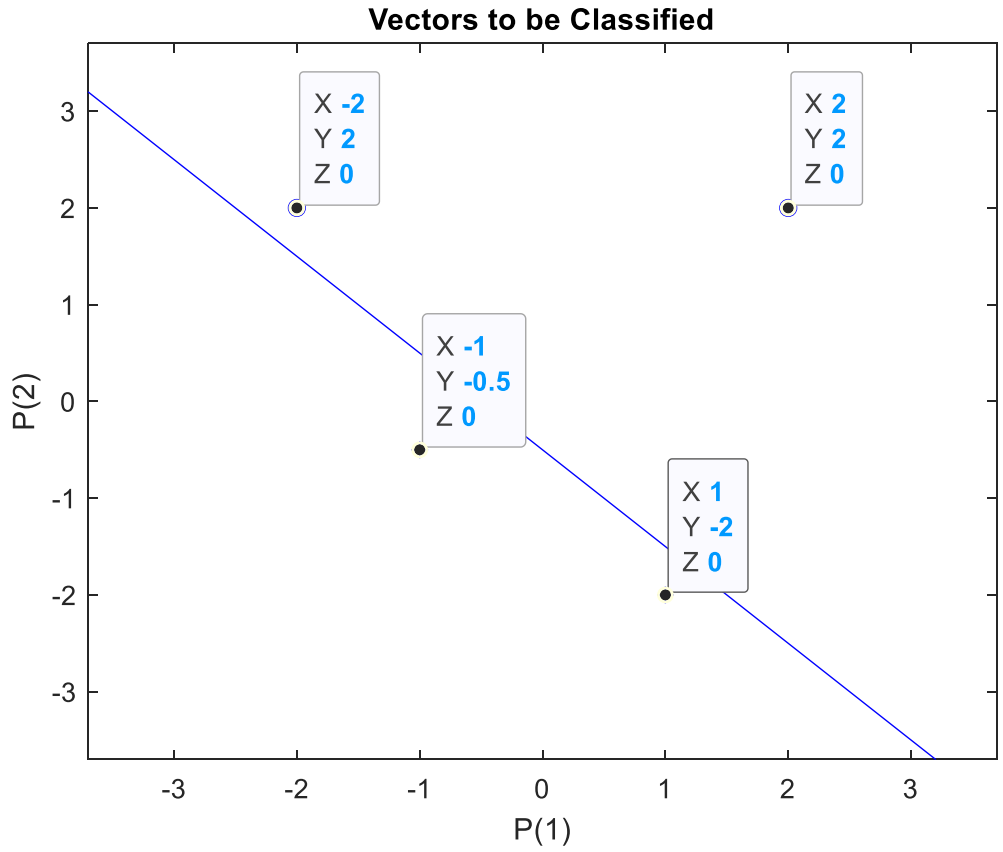


Рисунок 3.9 - Разделяющая прямая персептрона

```
>> plotpc(net.IW{1,1},net.b{1});
```

Якість якості мережі можна виконати за допомогою команди sim:

```
>> a = sim(net,p)
a =
0 1 0 1
```

Вихід мережі збігається з цільовим вектором *t*.

Наступний приклад описує підготовку персептрона реалізації функції OR:

```
>> percOR=newp([0 1; 0 1],1);
>> input = [0 0 1 1; 0 1 0 1]; d=[0 1 1 1];
>> plotpv(input,d);
>> percOR.adaptParam.passes=20;
>> percORa=adapt(percOR,input,d);
>> plotpc(percORa.IW{1},percORa.b{1});
>> rez=sim(percORa,[0 0 1 1; 0 1 0 1])
rez =
0 1 1 1
```

На рисунку 3.10 показує отриману лінію ділення.

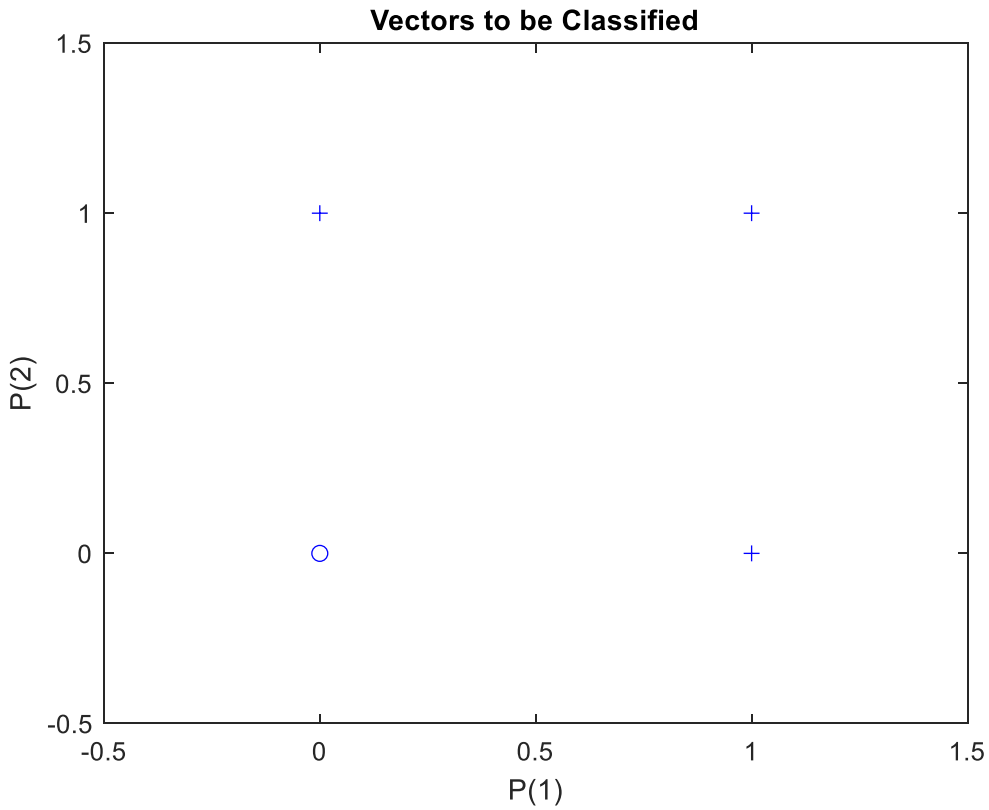


Рисунок 3.10 – Персептрона реалізація функції OR

Розглянемо більш складний приклад. Ми встановили на площині набір із чотирьох класів A, B, C, D. Кожен клас - це набір точок на площині. Заняття змінюються відносно один одного для забезпечення розділення:

```
>> A = [rand(1,20) - 0.6; rand(1,20) + 0.6];
>> B = [rand(1,20) + 0.6; rand(1,20) + 0.6];
>> C = [rand(1,20) + 0.6; rand(1,20) - 0.6];
>> D = [rand(1, 20) - 0.6; rand(1,20) - 0.6];
>> plot(A(1,:),A(2,:),'bs')
>> hold on
>> plot(B(1,:),B(2,:),'r+')
>> plot(C(1,:),C(2,:),'go')
>> plot(D(1,:),D(2,:),'m*')
>> grid on
>> text(-.1,1.7,'Class A')
>> text(1.1,1.7,'Class B')
>> text(-0.1,-0.7,'Class C')
>> text(1.1,-0.7,'Class D')
```

Отримані класи показані на рисунку 3.11.
Ми кодуємо кожен клас з двомовним кодом:

```
>> a = [0 1]';
>> b = [1 1]';
>> c = [1 0]';
>> d = [0 0]';
```

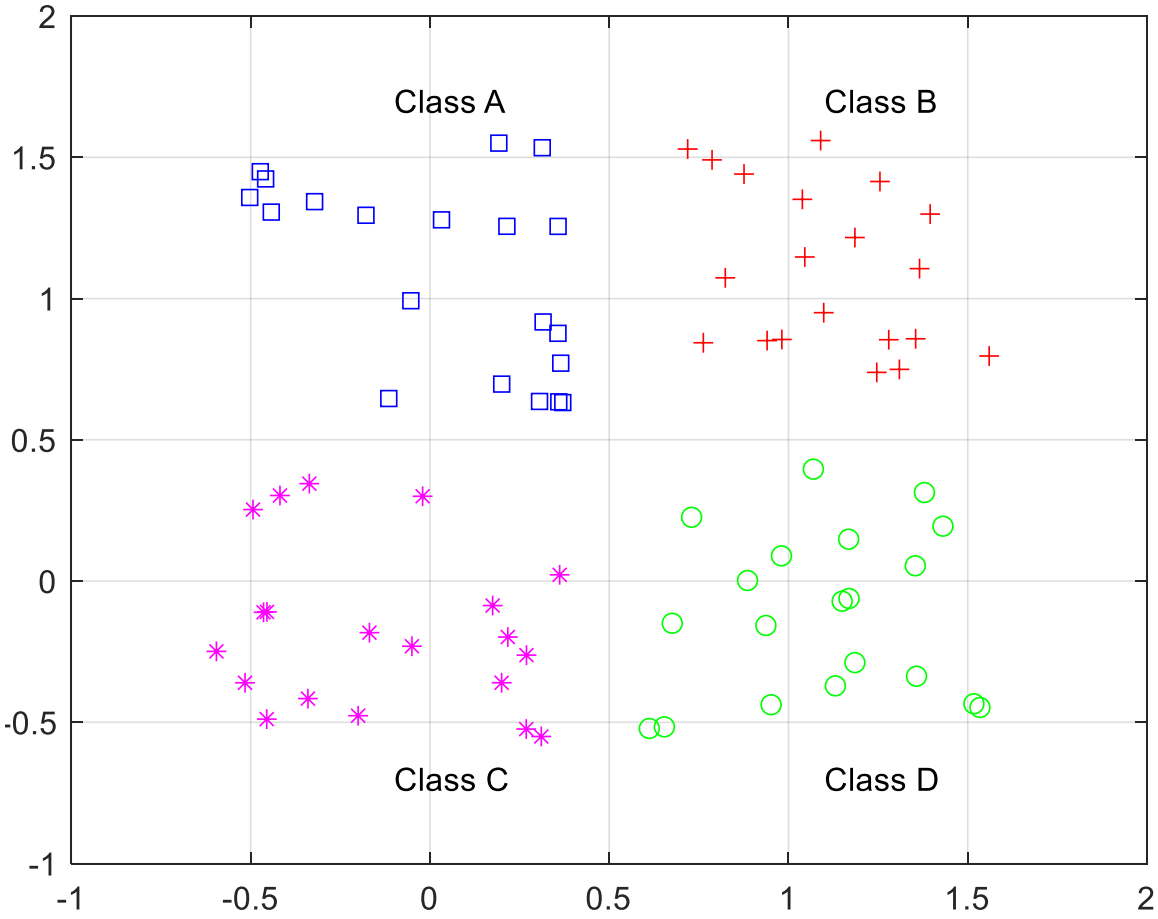


Рисунок 3.11 - Чотири класи об'єктів на площині

Задаємо вектор входу та цільовий вектор:

```
>> P = [A B C D];
>> T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
repmat(c,1,length(C)) repmat(d,1,length(D))];
```

Використовуючи останню команду, кожна точка площини ставить вектор у ряд. Тоді ми створюємо і викладаємо персептрон:

```
>> net = newp([-1 2;-1 2],2);
>> net = train(net,P,T);
```

Процес навчання персептрону ілюструє рисунок 3.12.

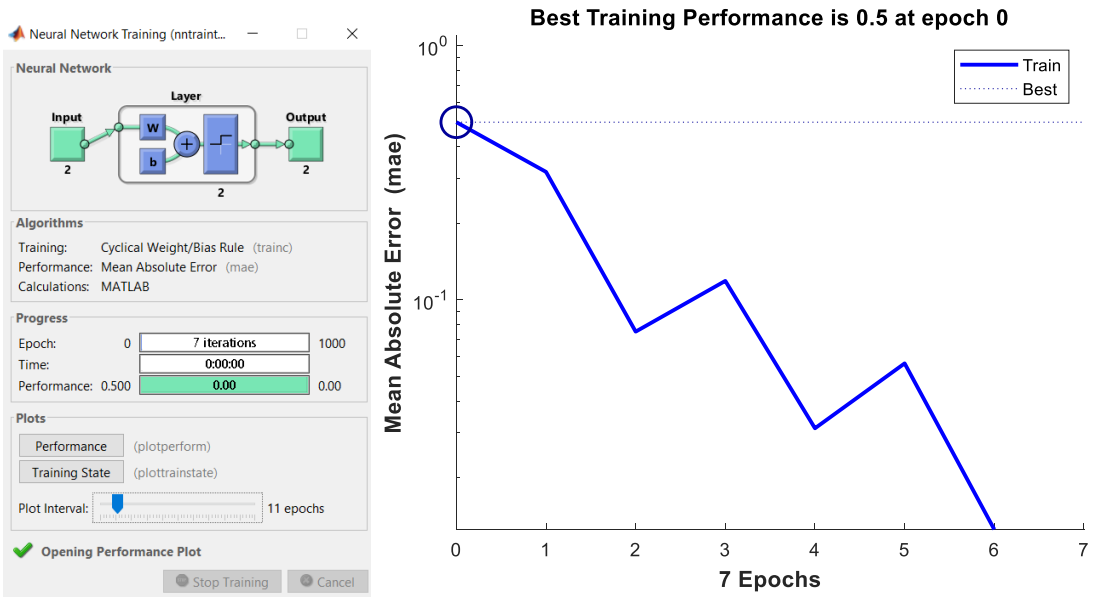


Рисунок 3.12 - Зміна помилок у навчальному процесі

Таким чином, у розглянутому прикладі персептрон має два бінарні результати.

3.3 Лінійна нейронна мережа

Лінійні нейронні мережі, або ADALIN (ADAptive Linear Neuron networks), за структурою аналогічні персептрону і відрізняються лише функцією активації та алгоритмом навчання.

Активаційна функція тут лінійна, а алгоритм навчання – метод найменших квадратів (МНК), який є ефективнішим, ніж правило навчання персептрону. Лінійний нейрон з двома входами наведено на рисунку 3.13.

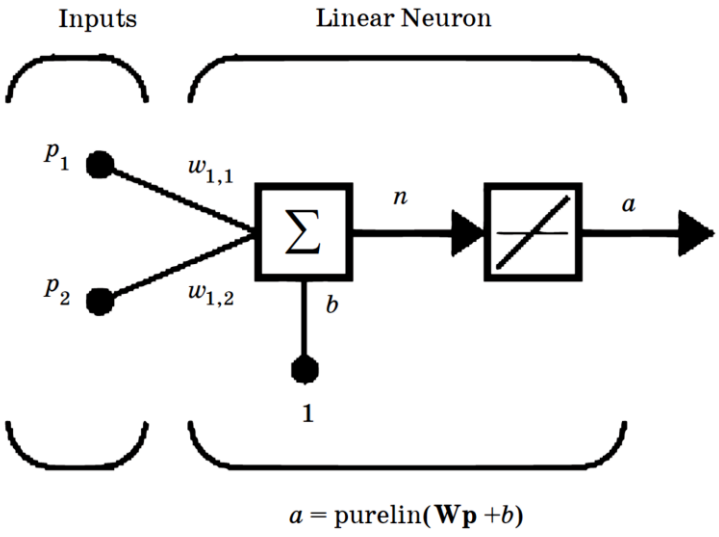


Рисунок 3.13 - Лінійний нейрон в MatLab

Як випливає з рис. 3.13 крім основних входів нейрони лінійної мережі мають вхід для постійного зміщення, рівного одиниці.

Лінійні мережі, як і персептрони, здатні вирішувати лише лінійно відокремлені задачі класифікації, що розділяє пряма визначається рівнянням

$$WP+b=0.$$

Можна записати

$$n = WP + b = w_{11}p_1 + w_{12}p_2 + b = 0.$$

Перетину з осями координат:

$$p_1 = 0: p_2 = -b/w_{12},$$
$$p_1 = 0: p_1 = -b/w_{12}.$$

Таким чином, вектор ваги тут визначає нахил прямий, а константа b - її зміщення щодо початку координат.

При $b = 0$ виконується $WP = 0$, тому вектор ваг завжди перпендикулярний роздільній прямій.

Розглянемо приклад.

Нехай $w_{1,1} = 1$, $w_{1,2} = 0{,}5$, $b = -1$. Тоді, наприклад, для точки з координатами $p_1 = 2$, $p_2 = 2$ маємо

$$a = purelin(WP + b) = [1\ 0.5]\begin{bmatrix}2\\2\end{bmatrix} - 1 = 2 > 0.$$

Лінійні нейрони можуть утворювати одношарову лінійну нейронну мережу (див. рис. 3.14).

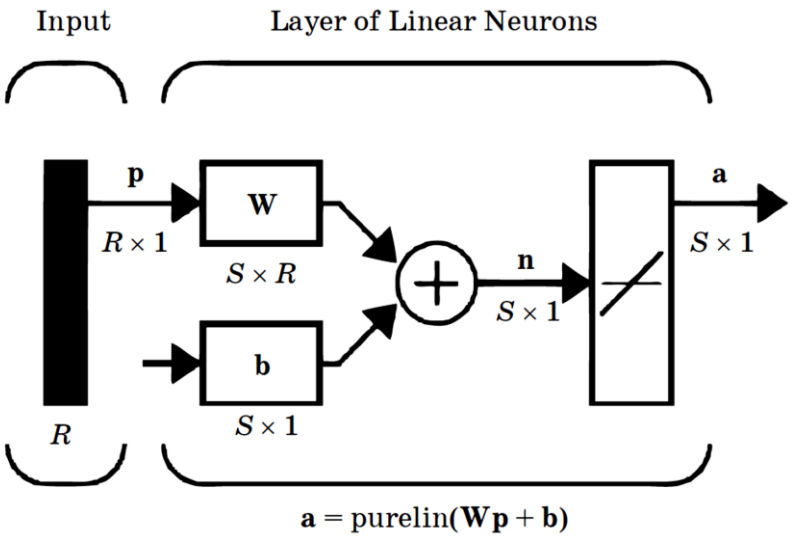


Рисунок 3.14 - Лінійна нейронна мережа в MatLab

Для створення лінійної НМ можна використовувати наступні варіанти команд:

>> net=newlin(PR, S, id, lr),

>> net=newlin(PR, S, 0, P),

>> net=newlind(P, T),

де *PR* - масив розміром R×2 мінімальних та максимальних значень *R* векторів входу; *S* – число нейронів; *id* - опис лінії затримки на вході мережі (за замовчуванням нуль); *lr* - параметр швидкості налаштування (за умовчанням 0,01); *P* - навчальні послідовності входів розміром *R*×*Q*; *Q* - число послідовностей; *T* - послідовність цілей для *P* розміром *S*×*Q*;

Розглянемо приклад.

Лінійна нейронна мережа створюється командою

>> net = newlin([-1 1; -1 1],1);

Тут матриця описує межі зміни двох скалярних входів, цифра «1» відповідає числу виходів, тобто числу нейронів. Задати значення ваги та зміщення можна командами

>> net.IW{1,1} = [1 0.5];

>> net.b{1} =[-1];

Вхідний вектор

>> p = [2; 2];

Моделювання роботи НМ

>> a = sim(net,p)

>> ADALIN

a =

2

3.4 Рекурентний метод найменших квадратів

Розглянемо використання методу найменших квадратів для навчання лінійних мереж.

Як і для персептрона, для лінійної мережі застосовується процедура навчання з учителем, яка використовує навчальну множину з *N* пар векторів:

$$\{P_1, Z_1\}, \{P_2, Z_2\}, \cdots, \{P_N, Z_N\},$$

де P_i - вхідний вектор; *Z*i – заданий вихідний вектор.

Нехай A_i – вихід мережі для входу P_i. Тоді функцію помилки на *k*-й ітерації можна у вигляді

$$E(k) = \frac{1}{N}\sum_{i=1}^{N}(Z_i - A_i)^2.$$

Уявимо лінійний нейрон на *k*-й ітерації у вигляді, наведеному на рис. 3.15.

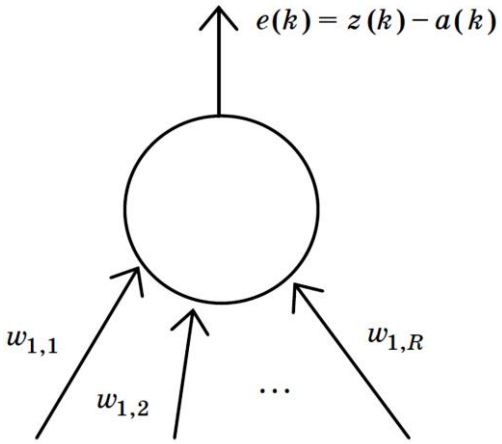


Рисунок 3.15 - Навчання лінійного нейрона

У процесі навчання мережі потрібно знайти такі значення ваги і зсувів, щоб значення *e* було мінімальним. Це завдання можна розв'язати, оскільки для лінійної мережі поверхня помилки як функція входів має єдиний мінімум. Тому функція помилки зменшується у напрямку, зворотному до її градієнта.:

$$\Delta w_{1,j} = w_{1,j}(k+1) - w_{1,j}(k) = -\alpha \frac{\partial\left(e^2(k)\right)}{\partial w_{1,j}}, \quad j = \overline{1,R},$$

$$\frac{\partial\left(e^2(k)\right)}{\partial w_{1,j}} = 2e(k)\frac{\partial e(k)}{\partial w_{1,j}} = 2e(k)\frac{\partial\left(z(k) - \alpha(k)\right)}{\partial w_{1,j}} =$$

$$= 2e(k)\frac{\partial\left(z(k) - \left(\sum_{j=1}^{R} w_{1,j}p_j(k) + b\right)\right)}{\partial w_{1,j}} = -2e(k)p_j(k), \qquad j = \overline{1,R}.$$

Отже,

$$\Delta w_{1,j} = w_{1,j}(k+1) - w_{1,j}(k) = -\alpha\left(-2e(k)p_j(k)\right) = 2\alpha e(k)p_j(k) =$$
$$= \eta e(k)p_j(k), \qquad j = \overline{1,R}.$$

$$w_{1,j}(k+1) = w_{1,j}(k) + \eta e(k)p_j(k), \qquad j = \overline{1,R}.$$

Аналогічно можна получить

$$b_j(k+1) = b_j(k) + \eta e(k), \qquad j = \overline{1,R}.$$

де n - константа швидкості навчання.

Таким чином, вага та усунення змінюються у напрямку антиградієнта пропорційна значенню помилки.

Для лінійної НМ з безлічі нейронів навчальні правила узагальнюються:

$$W(k+1) = W(k) + \eta E(k) P^T(k),$$
$$B(k+1) = B(k) + \eta E(k)/$$

Загалом алгоритм навчання передбачає такі кроки:

1. Вибирається перша пара з навчальної множини $\{P_1, Z_1\}$ (або випадкова пара).
2. Обчислюється помилка виходу мережі.
3. Коригуються ваги та усунення мережі.
4. Перевіряються критерії зупинки. Якщо вони не виконуються, вибирається наступна пара і повторюються кроки 2 і 3.

Як критерії закінчення процесу навчання може бути обраний поріг помилки e^2_{min} (по всій навчальній множині), або досягнення максимальної кількості ітерацій.

Розглянемо приклад. Нехай задано навчальну множину (N = 2):
$\{p_1 = [-1\ 1\ -1]^T, t_1 = [-1]\}$, $\{p_2 = [1\ -\ -1]^T, t_1 = [1]\}$.

Параметри алгоритму МНК: η= 0,4, $e^2_{min} = 0{,}1$.

На першій ітерації ваги дорівнюють нулю, і для першої навчальної пари

$$a(1) = W(1)P_1 = [0\ 0\ 0]\begin{bmatrix}-1\\1\\-1\end{bmatrix} = 0,$$
$$e(1) = t_1 - a(1) = -1 - 0 = -1,$$
$$W(2) = W(1) + \eta e(1)P^T(1) = [0\ 0\ 0] + 0.4(-1)\begin{bmatrix}-1\\1\\-1\end{bmatrix} = [0{,}4\ -0{,}4\ \ 0{,}4].$$

На другій ітерації розглядається друга навчальна пара:

$$a(2) = W(2)P_2 = [0{,}4\ -0{,}4\ \ 0{,}4]\begin{bmatrix}1\\1\\-1\end{bmatrix} = -0{,}4,$$
$$e(2) = t_2 - a(2) = -1 - (-0{,}4) = 1{,}4,$$
$$W(3) = W(2) + \eta e(2)P^T(2) = [0{,}4\ -0{,}4\ \ 0{,}4] + 0.4(1{,}4)\begin{bmatrix}1\\1\\-1\end{bmatrix} =$$
$$= [0{,}96\ 0{,}16\ \ -0{,}16].$$

Обчислювана помилка по всій навчальній множині дорівнює

$$W(3)_{p1} = W(3)P_1 = [0{,}96\ 0{,}16\ -0{,}16]\begin{bmatrix}-1\\1\\-1\end{bmatrix} = -0{,}64,$$

$$W(3)_{p2} = W(3)P_2 = [0{,}96\ 0{,}16\ -0{,}16]\begin{bmatrix}1\\1\\-1\end{bmatrix} = 1{,}28,$$

$$e^2 = \frac{1}{2}\left(t_1 - W(3)_{p1}\right)^2 + \left(t_2 - W(3)_{p2}\right)^2 =$$

$$= \frac{1}{2}((-1) - (-0{,}64))^2 + (1 - 1{,}25)^2 = 0{,}104.$$

Оскільки $e^2 = 0{,}104 > e'^2_{min} = 0.1$, ітерації мають бути продовжені. Розглянемо реалізацію цього простого прикладу в MatLab:

```
p=[-1 1; 1 1; -1 -1];
t=[-1 1];
net=newlin([p],1);
net.trainParam.goal=0.1;
[net,tr]=train(net,p,t);
```

Згідно результатам математичного моделювання наведеного на рисунку 3.16 робота закінчено після досягнення заданої кількості епох (57), необхідну якість навчання досягнуто.

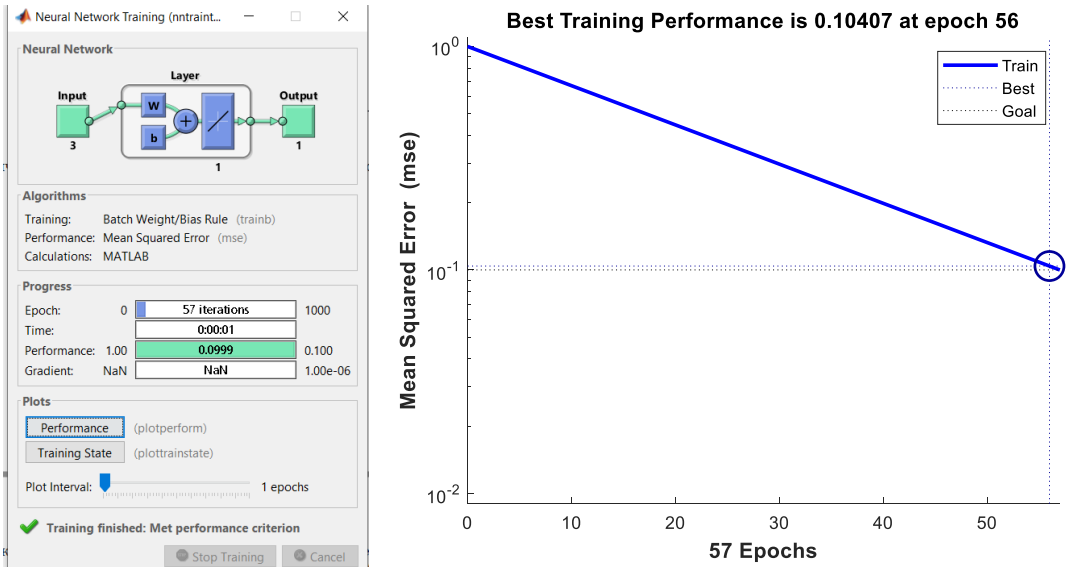


Рисунок 3.16 - Результати навчання лінійної НМ рекурентним методом найменших квадратів

Зауважимо, що з лінійної нейронної мережі справедливі самі обмеження, як і для персептрона, тобто навчальні дані мають бути лінійно роздільні.

Під час навчання ШНМ за допомогою процедури *train* можна вказувати точність (помилку) навчання та кількість епох навчання.

Наприклад,

```
p = [2 1 -2 -1;2 -2 2 1]; t = [0 1 0 1];
net = newlin( [-2 2; -2 2],1);
net.trainParam.goal= 0.1;
net.trainParam.epochs = 60;
net = train(net,p,t);
```

На рисунку 3.17 показано, як змінюється середньоквадратична помилка у процесі навчання, наближаючись до заданого порога.

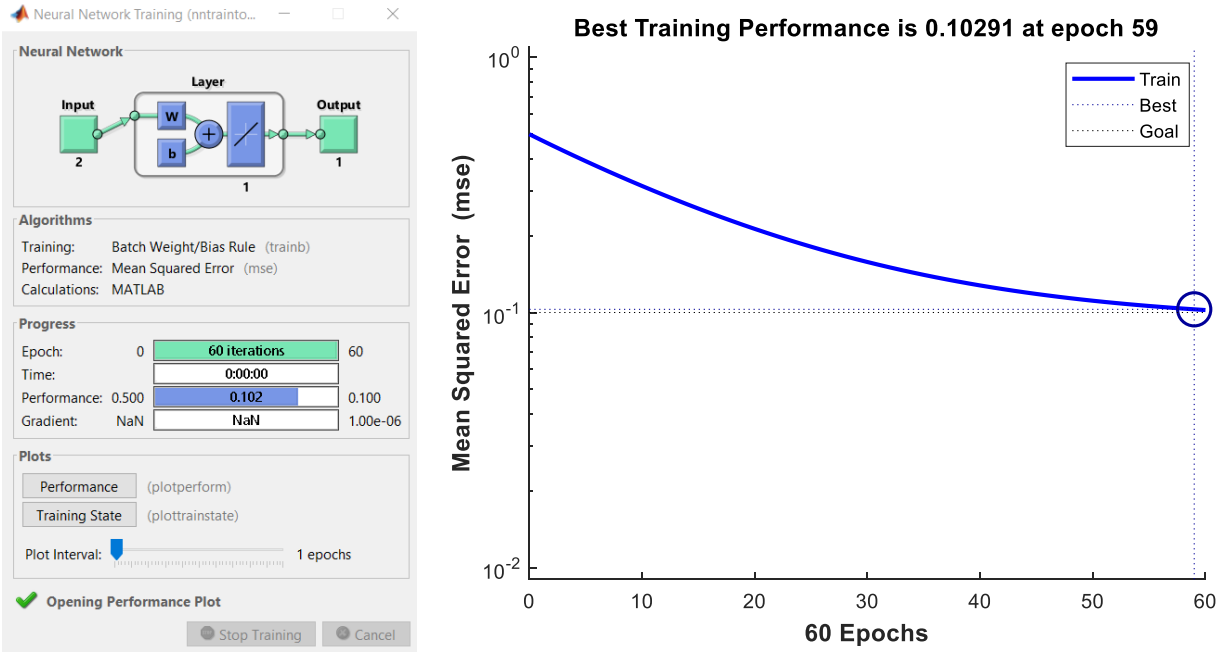


Рисунок 3.17 – Графік зміни середньоквадратичної помилки у процесі навчання, до заданого порога

Багатошарові лінійні мережі не мають сенсу, тому що подібна мережа завжди перетворюється на одношарову. Для підтвердження цієї тези розглянемо приклад лінійної нейронної мережі, наведений на рис. 3.18.

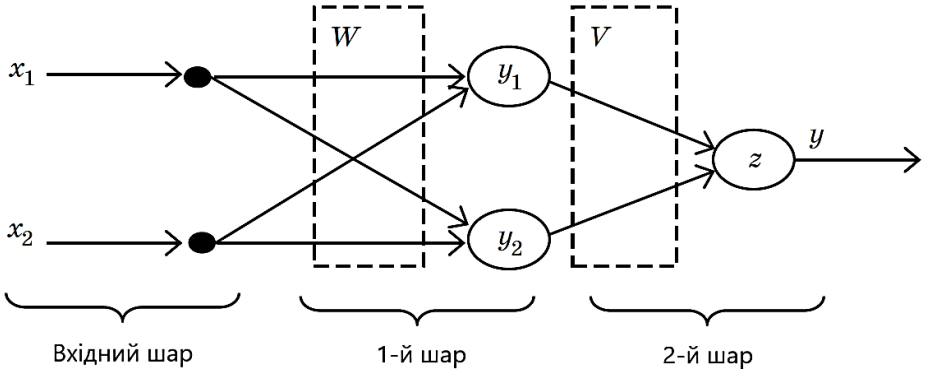


Рисунок 3.18 - Двошарова лінійна нейронна мережа

Можна записати

$$
\begin{gathered}
y_1 = x_1 w_{11} + x_2 w_{21} + b_1, \\
y_2 = x_1 w_{12} + x_2 w_{22} + b_2, \\
z_1 = y_1 v_1 + y_2 v_2 + b_3 = (x_1 w_{11} + x_2 w_{21} + b_1) v_1 + \\
+ (x_1 w_{12} + x_2 w_{22} + b_2) v_2 + b_3 \\
= x_1 (w_{11} v_1 + w_{12} v_2) + x_2 (w_{21} v_1 + w_{22} v_2) + b_1 v_1 + b_2 v_2 + b_3 = \\
= x_1 q_{11} + x_2 q_{21} + b.
\end{gathered}
$$

де $q_{11} = w_{11} v_1 + w_{12} v_2;\ q_2 = w_{21} v_1 + w_{22} v_2;\ b = b_1 v_1 + b_2 v_2 + b_3.$

При нульових зміщеннях можна використовувати матричний запис

$$Z = VW^T X = QX.$$

Комбінація з кількох нейронів ADALIN називається MADALINE (Many ADAptive LInear NEurons). Приклад наведено на рисунку 3.19.

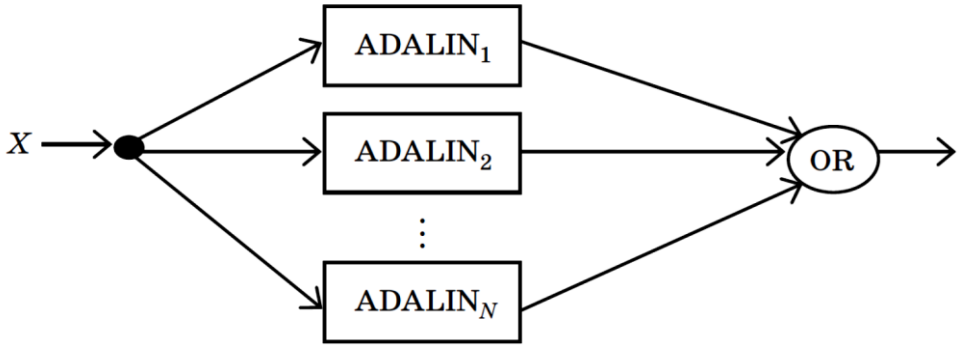


Рисунок 3.19. Приклад нейронної мережі MADALINE

Така ШНМ може виконувати, наприклад, розпізнавання символів незалежно від орієнтації на площині. Окремі модулі ADALIN відповідають за повороти символу на певний кут, а вузол OR забезпечує інваріантність розпізнавання.

3.5 Лінійна мережа з лінією затримки

Лінійні мережі із затримкою вхідного сигналу дозволяють створювати динамічні нейронні мережі ADALIN, які застосовуються при вирішенні завдань обробки сигналів та в системах керування.

Послідовність значень вхідного сигналу {p(k)} надходить на лінію затримки, що складається з блоку N-1 запізнення. Розглянемо приклад. Визначимо лінійну нейронну мережу:

```
>> net = newlin([0,11],1);
```

Опишемо дві лінії затримки на вході НМ (див. рис. 3.20):

```
>> net.inputWeights{1,1}.delays = [0 1 2];
```

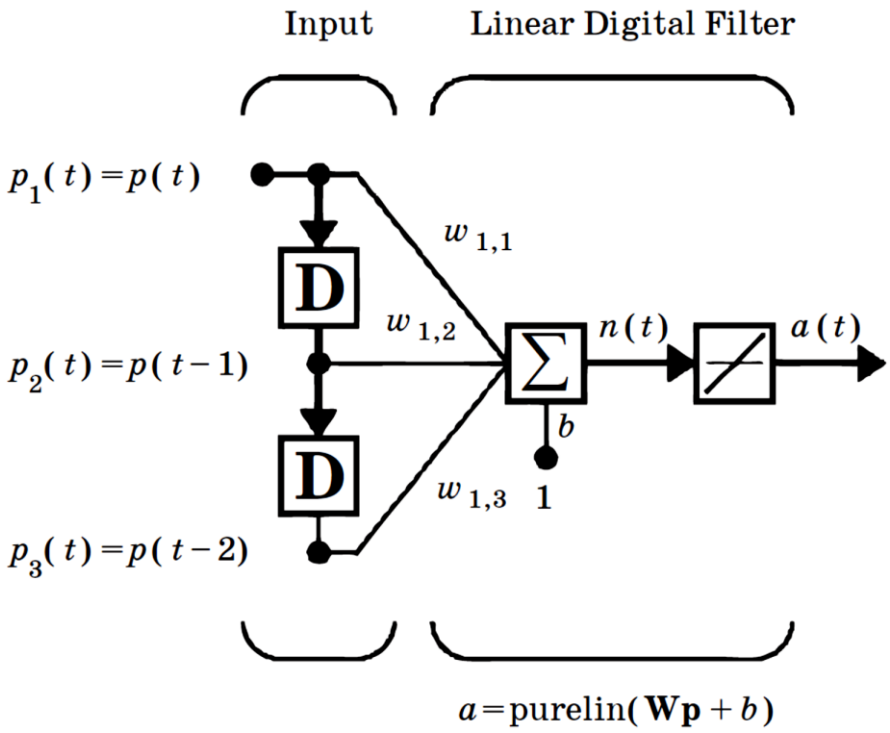


Рисунок 3.20 - Лінійна нейронна мережа з двома лініями затримки

Задаємо ваги та зміщення нейронної мережі

```
>> net.IW{1,1} = [2 4 11];
>> net.b{1} = [3];
```

та початкові значення виходів лінії затримки

```
>> pi ={1 2}
```

Вихід мережі описуватимемо формулою

$$a(k) = pureline(Wp + b) = \sum_{i=1}^{3} w_{1,i}p(k - i + 1) + b.$$

Розглянемо реакцію мережі під час подання деякої вхідної послідовності:

```
>> p = {1 2 5 9}
>> [a,pf] = sim(net,p,pi);
```

Вихідна послідовність має вигляд

```
>> ADALINtime

a =
  1×4 cell array
    {[24]}    {[33]}    {[32]}    {[63]}

pf =
  1×2 cell array
    {[5]}    {[9]}
```

Перевірка:

$$\begin{aligned} a(1) &= 1 \cdot 2 + 2 \cdot 4 + 1 \cdot 11 + 3 = 24, \\ a(2) &= 2 \cdot 2 + 1 \cdot 4 + 2 \cdot 11 + 3 = 33, \\ a(3) &= 5 \cdot 2 + 2 \cdot 4 + 1 \cdot 11 + 3 = 32, \\ a(4) &= 9 \cdot 2 + 5 \cdot 4 + 2 \cdot 11 + 3 = 63. \end{aligned}$$

Припустимо, що нейронна мережа повинна видавати у відповідь на вхідну послідовність 4, 5, 6, 7 задану вихідну послідовність 20, 25, 30, 35. Для цього потрібна адаптація ваг:

```
>> net = newlin([0,10],1);
>> net.inputWeights{1,1}.delays = [0 1 2];
>> net.IW{1,1} = [7 8 9];
>> net.b{1} = [0];
>> pi ={1 2};
>> p = {4 5 6 7};
>> T = {20 25 30 35}
>> net.adaptParam.passes = 200;
>> [net,y,E pf,af] = adapt(net,p,T,pi);

>> ADALINtime2

T =
  1×4 cell array
    {[20]}    {[25]}    {[30]}    {[35]}
```

Наступний приклад описує використання лінійної НС із затримкою для моделювання коливальної динамічної ланки з одиничним коефіцієнтом передачі та постійними часу рівними 1 с, яка наведена на рисунку 3.21, а результати моделювання перехідного процесу при подачі одиничного керуючого сигналу наведені на рисунку 3.22.

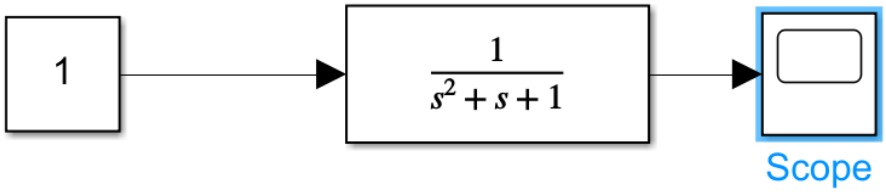


Рисунок 3.21 – Математична модель коливальної динамічної ланки в пакеті Simulink MatLab

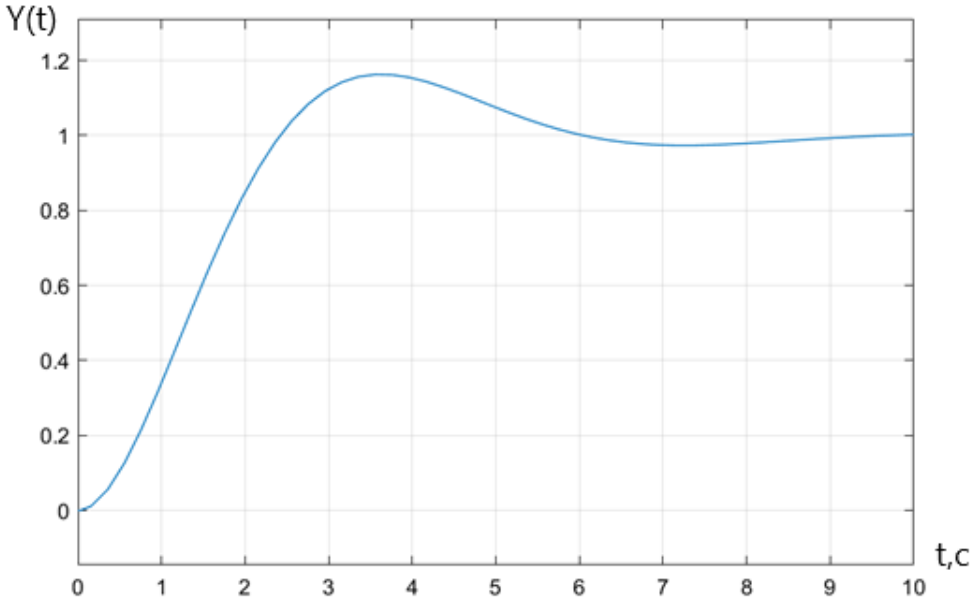


Рисунок 3.22 – Графік перехідного процесу коливальної ланки

Проведемо програмне моделювання коливальної динамічної ланки програмними засобами *MatLab*:

```
>> sys = ss (tf (1, [1 1 1])); % - опис коливальної ланки
>> t = 0:0.2:10; % - інтервал та крок моделювання
>> [Y, t] = step(sys, t) % - реакція ланки на стрибок
>> plot(t,Y)
>> grid
>> xlabel('t')
>> ylabel('Y(t)')
```

Наступні команди готують вихідні дані та виконують адаптацію ваг ШНМ:

```
>> P = Y(1: length(t)-2)';
>> T = Y(3: length(t))';
>> Z= [1 2];
>> net = newlin([-1 1], 1, Z);
>> pi ={0 0};
>> net.IW{1,1} = [1 1];
```

```
>> net.adaptParam.passes = 200;
>> P1 = num2cell(P);
>> T1 = num2cell(T);
>> [net,y,E pf,af] = adapt(net,P1,T1,pi);
```

Виконаємо моделювання ШНМ та порівняємо заданий та отриманий процеси (див. рис. 3.23):

```
>> x = sim(net,P1)
>> x1 = cat(1,x{:})';
>> t1=t(1: length(t)-2)';
>> plot(t1,x1,'b:+', t1,P,'r-o')
>> xlabel('t');
>> ylabel('Y(t)');
>> grid on
```

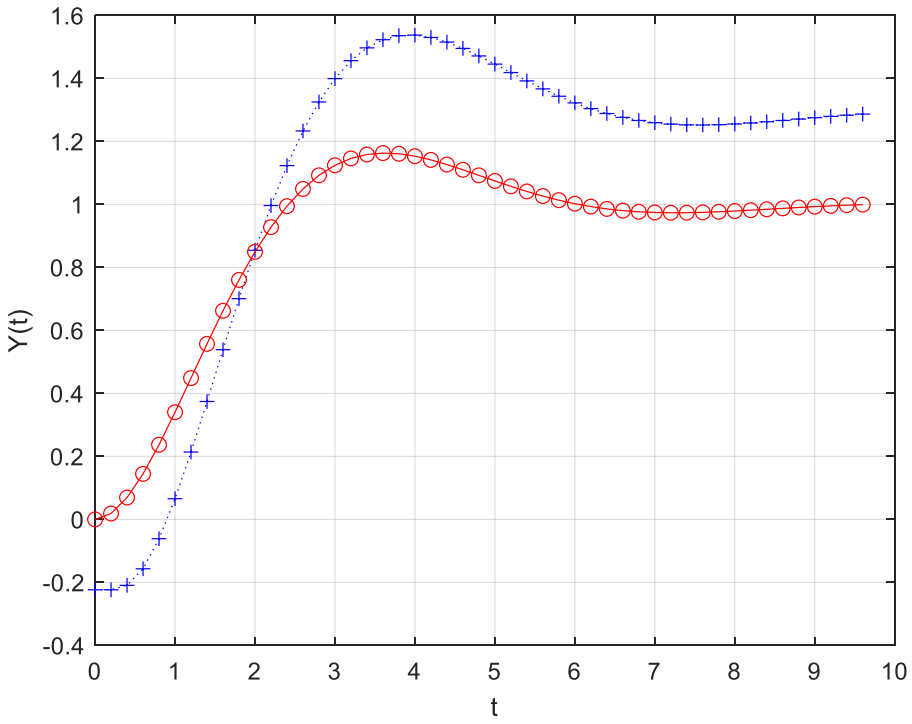


Рисунок 3.23 – Порівняльні графік перехідного процесу коливальної ланки

Помилка на початку перехідного процесу викликана методом формування навчальної вибірки.

При навчанні можна було б використати команду

```
>> [net,y,E pf,af] = train(net,P1,P1,pi);
```

Аналіз результатів моделювання при використанні команди train показує, що помилка початку перехідного процесу зменшилася (див. рис. 3.24). Результати навчання лінійної з затримкою НМ наведені на рисунку 3.25.

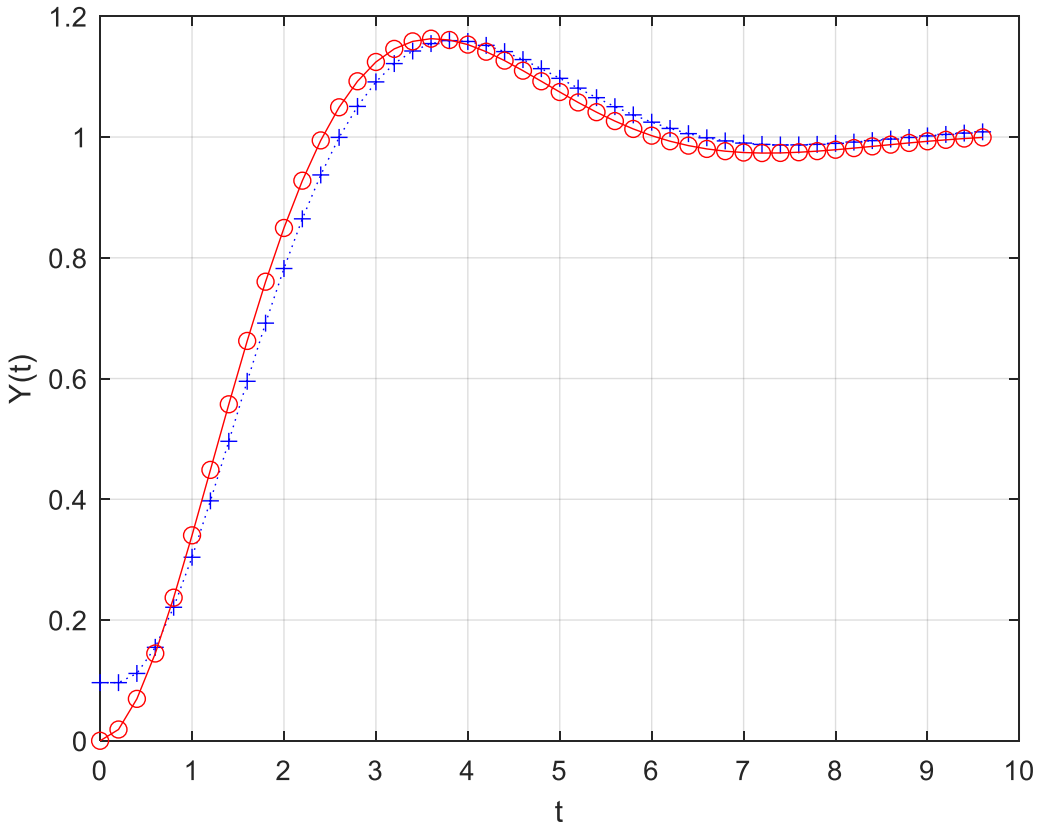


Рисунок 3.24 - Перехідні процеси після зміни навчальної вибірки

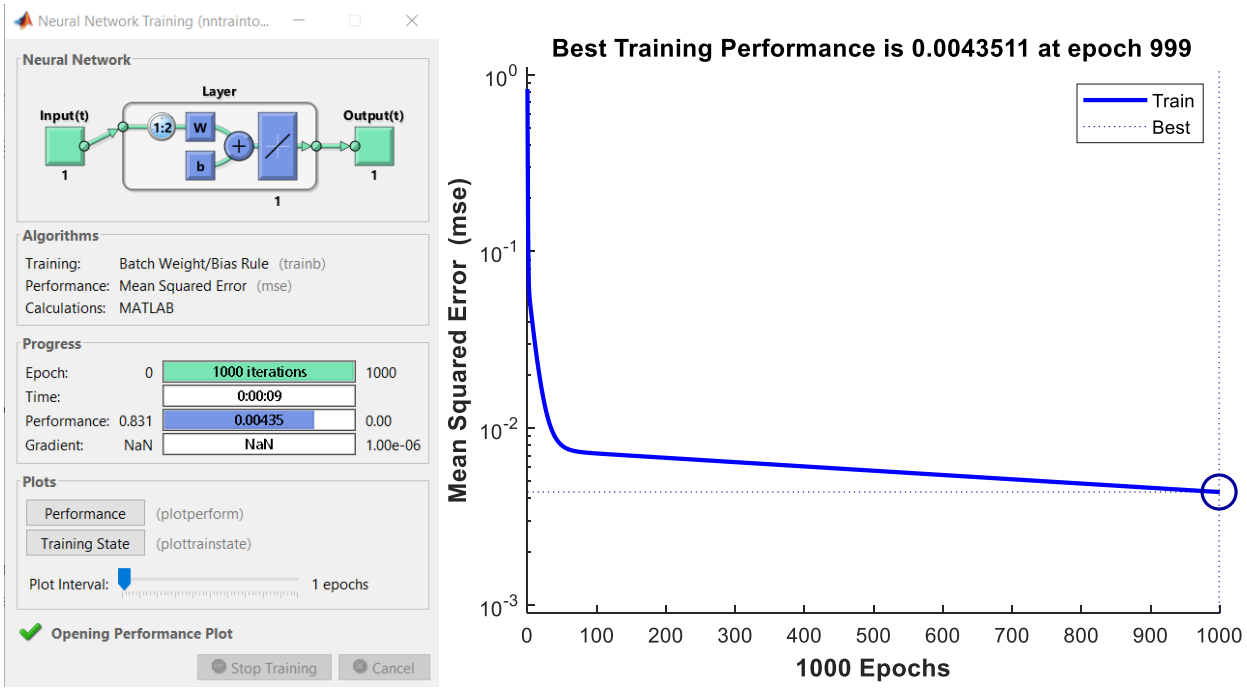


Рисунок 3.25 - Результати навчання лінійної НМ з затримкою

Розглянемо наступний приклад. Нехай потрібно, щоб лінійна ШНМ відтворювала сигнал, наведений на рис. 3.26. Йому відповідає набір команд

```
>> t = 0: 0.01: 6;
>> y = [sin(4.1*pi*0.2.*t.*abs(0.7*t - 5))];
>> plot(t,y)
>> ylim([-1.2 1.2])
>> xlabel(' Time [sec]');
>> ylabel(' Target Signal');
>> grid on
```

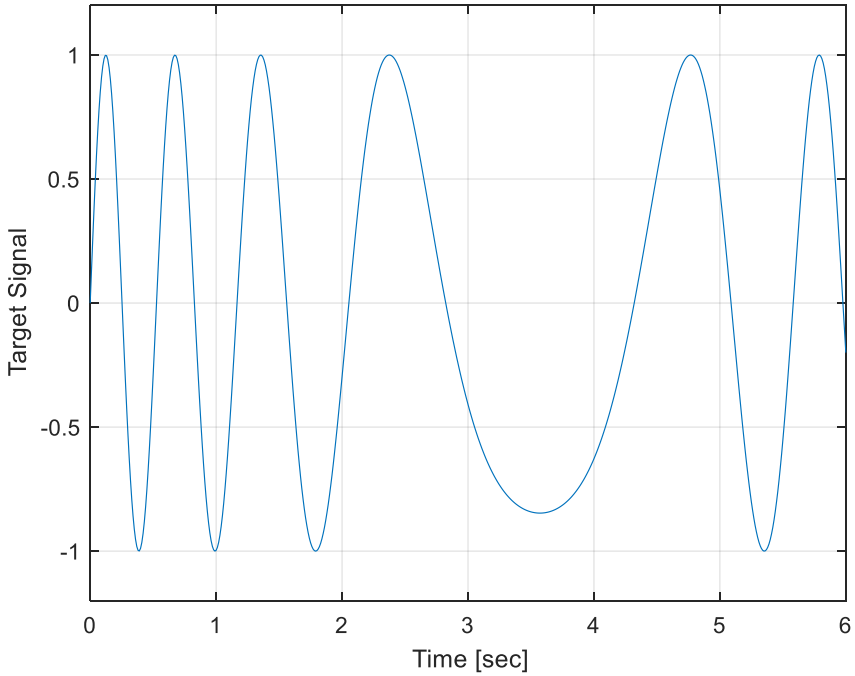


Рисунок 3.26 - Цільовий сигнал для лінійної ШНМ із затримкою

Визначимо конфігурацію ШНМ та навчимо її:

```
>> p = con2seq(y);
>> net = newlin([-1,1],1);
>> net.inputWeights{1,1}.delays = [0 1 2 3 4 5];
>> [net,Y,E] = adapt(net,p,p);
```

Розглянемо помилку передбачення ШНМ:

```
>> E = seq2con(E); E = E{1};
>> plot(t,E);
>> grid on
>> legend('Prediction error')
>> xlabel('Time [sec]');
>> ylabel('Error');
>> ylim([-1.2 1.2])
```

Графік помилки подано на рис. 3.27. Найбільші значення помилки припадають на початок перехідного процесу.

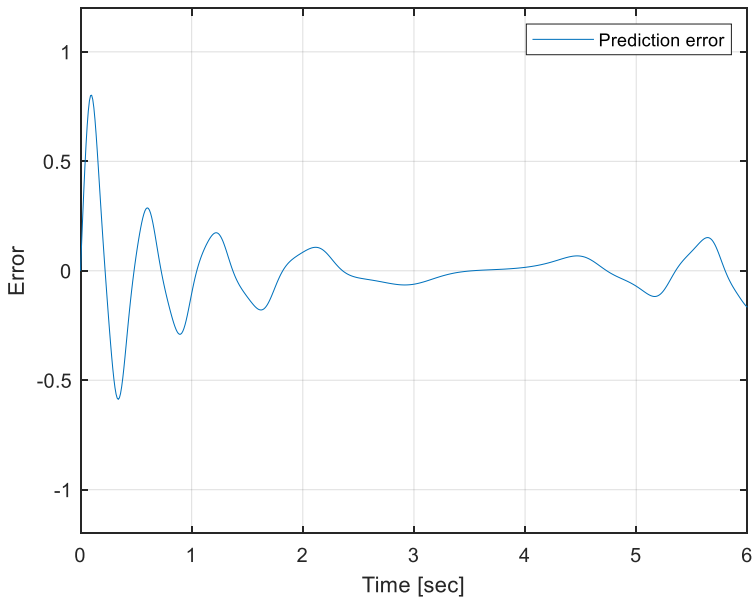


Рисунок 3.27 - Помилка передбачення лінійної ШНМ із затримкою

Порівнюємо результати моделювання ШНМ (див. рис. 3.28)

```
>> plot(t,y-E, t,y); %результат відновлювання сигналу
>> grid on
>> legend('Prediction error')
>> xlabel('Time [sec]');
>> ylabel('Y(t)');
>> ylim([-1.2 1.2]);
```

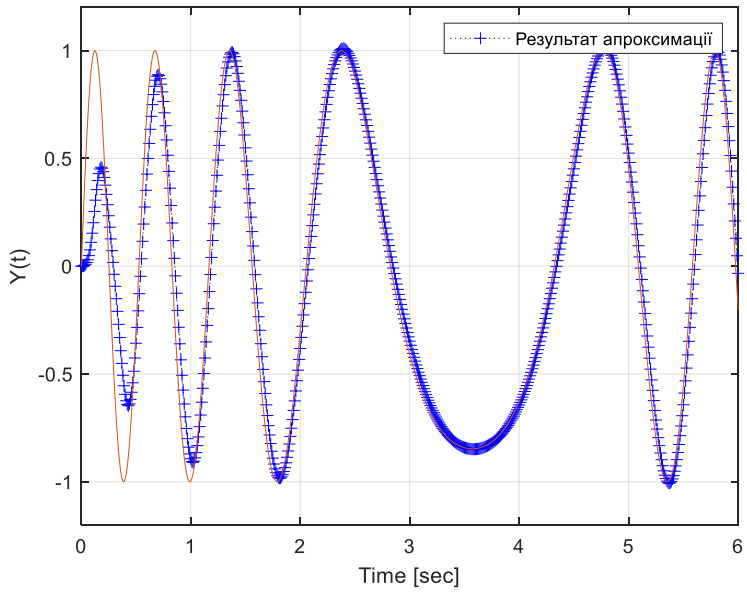


Рисунок 3.28 – Порівняння графіків цільового сигналу з графіком апроксимації яка отримана ШНМ

Після навчання були отримані наступні ваги ШНМ:

```
>> net.IW{1}
ans =
2330 0.2045 0.1735 0.1405 0.1059 0.0702
```

4 НЕЙРОНІ МЕРЕЖІ ПРЯМОГО ПОШИРЕННЯ

4.1 Топологія та властивості

Штучна нейронна мережа прямого поширення (ШНМ ПП) (англ. feed-forward network) містить один або багато шарів нейронів. Такі мережі іноді називають *багатошаровим персептроном* (multi-layer perceptron).

Можна виділити такі властивості ШНМ ПП:

- між нейронами всередині одного шару відсутні зв'язки;
- відсутні зворотні зв'язки між шарами;
- нейрон наступного шару отримує сигнали від усіх нейронів попереднього шару (повнозв'язність);
- число входів, виходів, і навіть число нейронів у внутрішніх шарах необов'язково однаково.

Мережі прямого поширення немає зворотних зв’язків (т. е. немає пам'яті). Цим вони нагадують комбінаційні логічні схеми, оскільки сигнал на виході повністю визначається поточними входами, вагами зв'язків та активаційними функціями нейронів.

Більшість нейронних мереж, що використовуються на практиці, відноситься до класу ШНМ ПП.

На рисунку 4.1 показаний варіант найпростішої одношарової ШНМ ПП з m нейронами та вхідним вектором довжиною n.

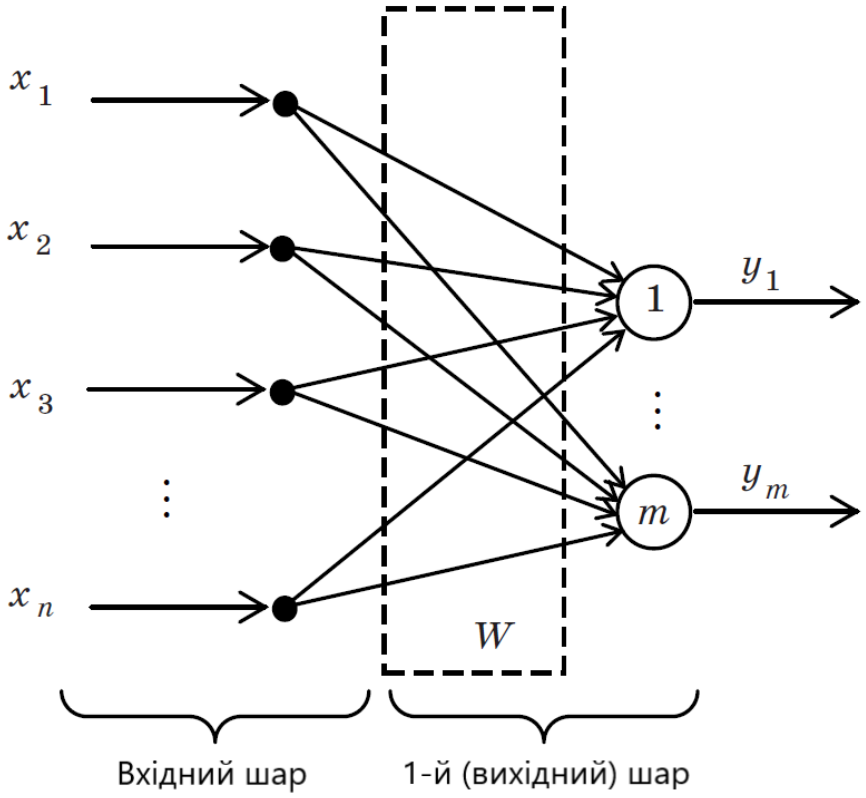


Рисунок 4.1 - Одношарова ШНМ ПП

Багатошарова НМ складається з множин нейронів і ваг, що чергуються. У цьому кожен шар ШНМ може мати довільне число нейронів. На рисунку 4.2 наведено тришарову мережу (вхідний шар просто розподіляє сигнали).

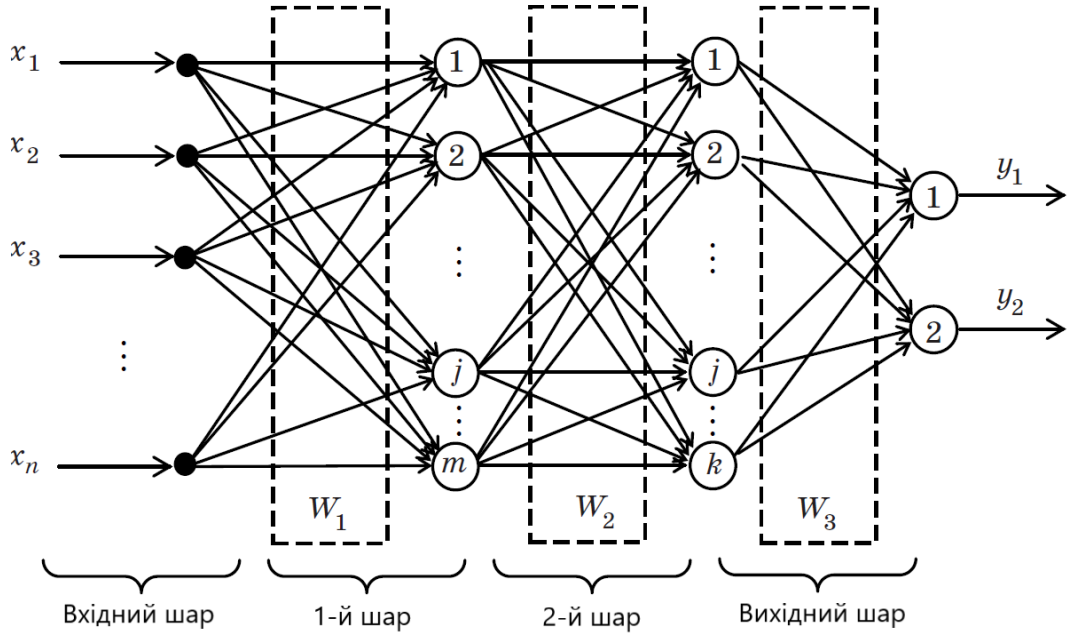


Рисунок 4.2 - Приклад тришаровий ШНМ ПР

Багатошарові ШНМ ПП є універсальними апроксиматорами - з їх допомогою можна описати будь-яку функцію від однієї чи безлічі змінних. Але для цього активаційні функції нейронів мережі мають бути нелінійними. У двошарових мережах часто вибирається активна сигмоїдна функція для нейронів 1-го шару, і лінійна - для нейронів 2-го шару.

Слід відмітити, що в загальному випадку не можна заздалегідь передбачити, скільки шарів і скільки нейронів повинна мати ШНМ ПП для вирішення конкретної проблеми. Це питання вирішується шляхом спроб і помилок з урахуванням наявного досвіду.

Якщо ШНМ ПП має два входи, то з її допомогою можна виконувати складні класифікації, виділяючи на площині опуклі або неопуклі, обмежені або необмежені області. Якщо ШНМ ПП має три входи, то вона виконує завдання класифікації у просторі, якщо більше трьох входів – то у гіперпросторі.

У системі MatLab ШНМ ПП створюється командою:

NEWFF(PR,[S1 S2...SN],{TF1 TF2...TFN},BTF,BLF,PF),

де *PR* - матриця *R*×2 мінімальних та максимальних значень для кожного з *R* вхідних елементів; S_1 , S_2, ..., S_N - число нейронів у кожному шарі (від 1-го до *N*-го); TF_1, TF_2, ..., TF_n - передавальні функції кожного

шару (за умовчанням 'tansig'); BTF, BLF - функції навчання (за я 'trainlm' і 'learngdm'), PF - функція оцінки якості роботи (за умовчанням 'mse').

Приклад 4.1. Створити двошарову ШНМ ПП з одним входом та одним виходом:

```
>> P = [0 1 2 3 4 5 6 7 8 9 10];
>> net = newff (minmax (P), [5 1], {'tansig' 'purelin'});
```

Приклад 4.2. Створити ШНМ ПР з двома входами та трьома нейронами у кожному з двох шарів:

```
>> netll = newff([-1 2; 0 5], [3, 3]);
```

4.2 Алгоритм зворотного розповсюдження помилки

Дельта-правило, яке застосовується під час навчання персептрона, використовує величину помилки вихідного шару. Якщо мережа має два або більше шарів, то для проміжних шарів значення помилки в явному вигляді не існує, і використовувати дельта-правило не можна.

Основна ідея зворотного поширення полягає в тому, як отримати оцінку помилки для прихованих нейронів шарів. Зауважимо, що ***відомі*** помилки, які допускаються нейронами вихідного шару, виникають внаслідок *невідомих* помилок нейронів прихованих шарів. Чим більше значення синаптичного зв'язку між нейроном прихованого шару і вихідним нейроном, тим більше помилка першого впливає на помилку другого. Отже, оцінку помилки елементів прихованих шарів можна отримати як виважену суму помилок наступних шарів.

Алгоритм зворотного поширення помилки (АЗПП), що є узагальненням дельта-правила, дозволяє навчати ШНМ ПП із будь-яким числом шарів. Можна сміливо сказати, що АЗПП фактично використовує різновид градієнтного спуску, перебудовуючи ваги у бік мінімуму помилки.

При використанні АЗПП передбачається, що використовується активаційна сигмоїдна функція. Ця функція дозволяє заощаджувати обчислювальні витрати, оскільки має просту похідну:

$$\frac{dF(y)}{dy} = F(y)(1 - F(y))$$

Сигмоїдна функція обмежує значенням одиниця «1» сильні сигнали і посилює слабкі.

Сенс алгоритму зворотного поширення помилки у тому, що при навчанні мережі спочатку пред'являється образ, для котрого обчислюється помилка виходу. Далі ця помилка поширюється по мережі у зворотному напрямку, змінюючи ваги міжнейронних зв'язків.

Алгоритм включає таку ж послідовність дій, як і при навчанні персептрона. Спочатку ваги міжнейронних зв'язків набувають випадкові значення, потім виконуються наступні кроки:

1) вибирається навчальна пара (X, Z^*), X подається на вхід;

2) обчислюється вихід мережі $Z = F(Y)$;

3) розраховується помилка виходу E;

4) ваги мережі коригуються з метою мінімізації помилки;

5) повернення до п. 1 і т. д., доки не буде мінімізована помилка по всіх навчальних парах.

Кроки 1 і 2 - це пряме поширення мережі, а кроки 3 і 4 - зворотне.

Перед навчанням необхідно розбити наявні пари вхід-вихід на дві частини: *навчальні* та *тестові*.

Тестові пари використовуються для перевірки якості навчання: нейрона мережа добре навчена, якщо при заданій тестовій парі видає на вході вихід, близький до тестового.

При навчанні можлива ситуація, коли НМ показує хороші результати для навчальних даних, і погані – для тестових. Це може бути обумовлено двома причинами:

1. Тестові дані сильно відрізняються від навчальних, тобто пари, що навчають, охоплювали не всі області вхідного простору.

2. Виникає явище «перенавчання» (overfitting), у якому поведінка НМ виявляється складнішим, ніж вирішуване завдання.

Останній випадок завдання апроксимації функції по точках ілюструє рисунок 4.3, на якому світлі кружки відповідають тестовим даним, а темні – навчальним.

На рисунку 4.3 суцільна лінія означає вихід НМ для довільного входу з області визначення X. Графік на рис. 4.3а відповідає добре навченої НМ, а графік на рис. 4.3,б - перенавченої.

Розглянемо для простоти викладу двошарову ШНМ ПП, в якій ваги прихованого шару описуються вектором W, а ваги вихідного шару - вектором V (див. рис. 4.4).

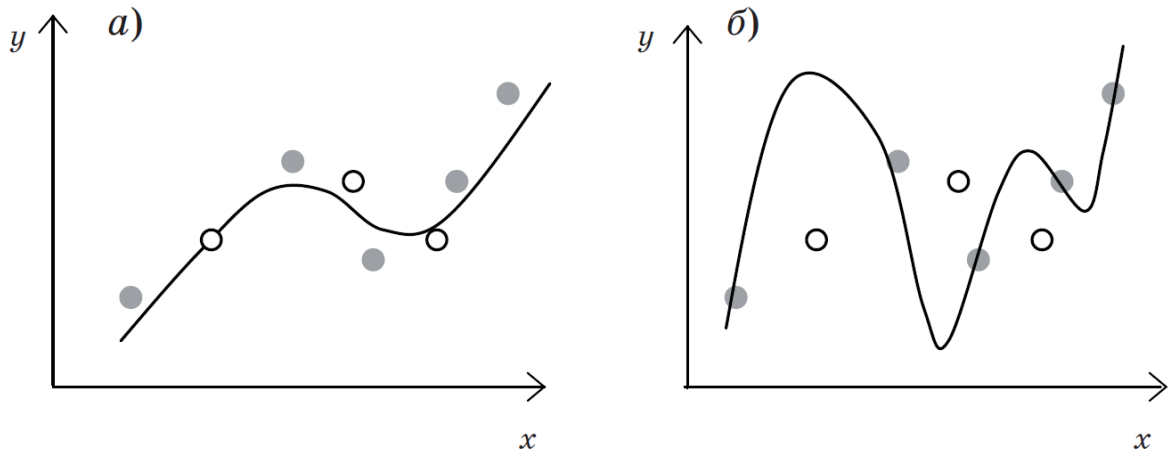


Рисунок 4.3 - Ілюстрація явища перенавчання

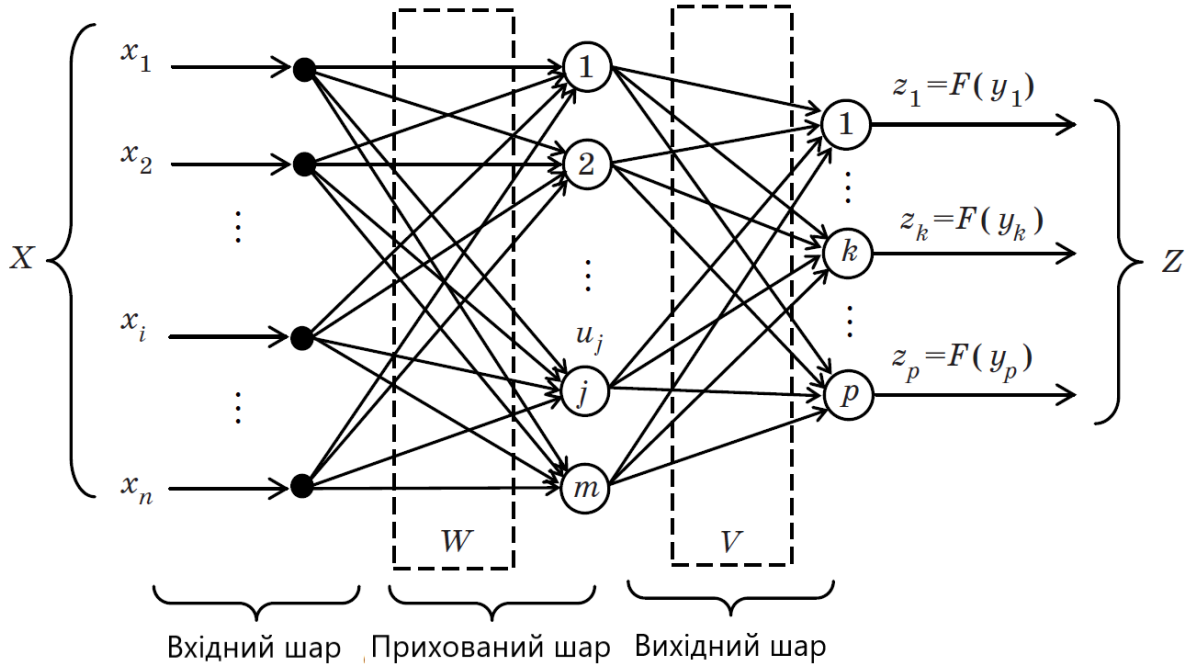


Рисунок 4.4 - Двошарова ШНМ ПП

Класичний градієнтний метод пошуку мінімуму функції *f*(*X*), де *X* - вектор, полягає у зміні аргументу у напрямку антиградієнта:

$$X(t+1) = X(t) - \eta \frac{\partial E(X)}{\partial X}.$$

Для вихідного шару можна записати

$$V(t+1) = V(t) - \eta \frac{\partial E(V)}{\partial V}.$$

Таким чином, вага v_{jk}, що зв'язує *j*-й нейрон прихованого шару та *k*-й нейрон вихідного, коригується за формулою

$$v_{jk}(t+1) = v_{jk}(t) - \eta \frac{\partial E}{\partial v_{jk}}.$$

Помилка виходу може бути описана таким чином:

$$E = \frac{1}{2}\sum_{k=1}^{p}(z_k - z_k^*)^2.$$

Очевидно, що *E* явно не залежить від *V*, але для отримання похідної можна використовувати правила диференціювання складної функціїд

$$\frac{\partial E}{\partial z_k} = z_k - z_k^* = \Delta_k,$$
$$\frac{\partial E}{\partial y_k} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial y_k} = \Delta_k z_k(1 - z_k),$$
$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial y_k}\frac{\partial y_k}{\partial v_{jk}} = \Delta_k z_k(1 - z_k)u_j,$$

де u_j - вихід j-го нейрона прихованого шару

$$y_k = \sum_{k=1}^{p} v_{jk}u_j.$$

Таким чином, формула для корекції ваг вихідного шару набуває вигляду.

$$v_{jk}(t+1) = v_{jk}(t) - \eta\Delta_k z_k(1 - z_k)u_j.$$

Розглянемо далі корекцію ваг прихованого шару. Тут використовується формула

$$w_{ij}(t+1) = w_{ij}(t) - \eta\frac{\partial E}{\partial w_{ij}}.$$

Вихід нейрона прихованого шару описується формулою

$$a_j = \sum_{i=1}^{n} w_{ij}x_i,$$
$$u_j = F(a_j)$$

За аналогією з формулою для вихідного шару можна записати

$$\frac{\partial E}{\partial a_j} = \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial a_j} = \Delta_j u_j(1 - u_j).$$

Однак величина помилки для прихованого шару Δ_j не задана, тобто не ясно, яким має бути еталонний вихід прихованого шару. У той же час очевидно, що помилки вихідного шару залежать від помилки прихованого шару ШНМ, і цей вплив тим більше, чим більша вага зв'язку між прихованим нейроном шару і вихідним нейроном. Таким чином, оцінку помилки нейрона прихованого шару можна отримати як виважену суму помилок вихідного шару. (див. рис. 4.5):

$$\frac{\partial E}{\partial u_j} = \Delta_j = \sum_{k=1}^{p} \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial u_j} = \sum_{k=1}^{p} \Delta_k z_k (1 - z_k) v_{jk},$$

$$\frac{\partial E}{\partial u_j} = \frac{\partial E}{\partial u_j}\frac{\partial u_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \left(\sum_{k=1}^{p} \Delta_k z_k (1 - z_k) v_{jk} \right) u_j (1 - u_j) x_i.$$

Тоді формула для корекції ваг прихованого шару остаточно набуває наступного вигляду:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \left(\sum_{k=1}^{p} \Delta_k z_k (1 - z_k) v_{jk} \right) u_j (1 - u_j) x_i.$$

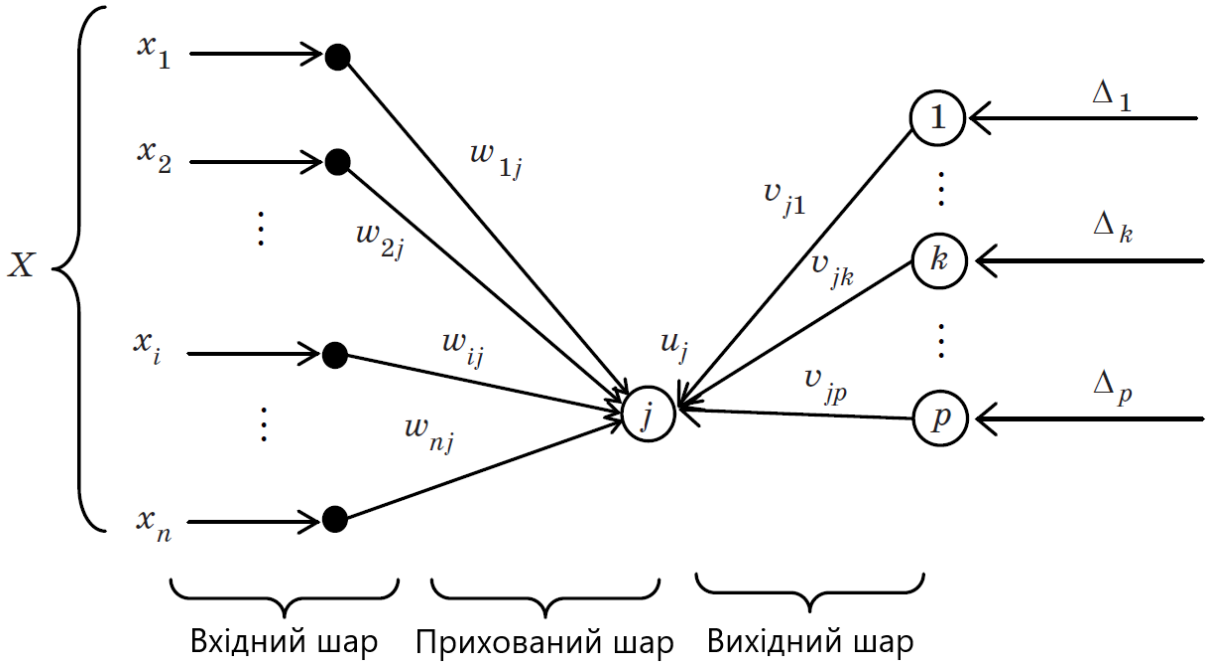


Рисунок 4.5 - Зворотне поширення помилки

Достоїнство АЗПП полягає в тому, що він узагальнюється для ШНМ ПП з будь-яким числом шарів.

Алгоритм зворотного поширення помилки знайшов застосування у низці прикладних розробок. Особливо вражаючі успіхи досягнуто при розпізнаванні друкованих літер, коли ефективність алгоритму близька до 100%. Однак застосування АЗПП в інших завданнях може бути утруднене через причини, характерні для градієнтних алгоритмів мінімізації:

- при великих значеннях сигмоїдної функції її похідна стає дуже малою, і навчання сповільнюється;
- поверхня помилки складної мережі може мати локальні мінімуми, виявивши які мережа перестане покращувати свою поведінку.

Роботу АЗПП можна поліпшити з огляду на другі похідні активаційної функції.

Прискорення роботи алгоритму можна досягти з використанням змінної швидкості навчання η. На початку роботи АЗПП її величина має значення, близьке до одиниці, та був послідовно зменшується приблизно 0,01. Це дозволяє швидко підійти до околиці мінімуму, а потім точно потрапити до нього.

Проте локальність АЗПП є важливим обмеженням використання. Якщо функція помилки має складний характер, потрібно використовувати методи випадкового пошуку. Проте АЗПП може бути успішно застосовано у багатьох практичних завданнях, деякі з яких будуть розглянуті далі.

4.3 Реалізація логічних функцій

Розглянемо завдання класифікації бінарних векторів довжиною 4 біти: якщо вектор містить непарне число одиничних бітів, то на виході мережі має бути 1, інакше - 0.

Приклад 4.3. Вхідні вектори описуються матрицею

```
>> inp = [0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1;
          0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1;
          0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1;
          0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1]
```

Вихідний вектор:

```
>> out=[0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0]
```

Створимо мережу прямого розповсюдження за допомогою команди

```
>> network=newff([0 1;0 1; 0 1; 0 1],[6 1],{'logsig','logsig'})
% функція активації - 'logsig' сигмоїдна
```

У цій команді перший масив містить діапазони зміни сигналу для кожного входу, другий масив - розмір кожного шару (кількість нейронів), потім задається тип передавальної функції для кожного шару.

До початку навчання виконується ініціалізація НМ:

```
>> network=init(network);
```

Задаємо кількість епох, швидкість і запустимо навчання:

```
>> network.trainParam.epochs = 500;
>> network.trainParam.lr = 0.05;
>> network=train(network,inp,out);
```

Перевіримо роботу мережі:

```
>> y=sim(network,inp);
>> out-y
```

```
ans =
   1.0e-03 *
  Columns 1 through 7
0.0014   0.0013   0.0014  -0.0010   0.0056  -0.0024  -0.0015
  Columns 8 through 14
0.0015   0.0012  -0.3317  -0.0013   0.0031  -0.0021   0.0218
  Columns 15 through 16
0.0085  -0.0048
```

Помилки між заданим виходом НМ і реальним виходом можна вважати дуже малими.

Результати навчання НМ класифікації бінарних векторів у прикладі 4.3 зображені на рисунку 4.6.

Розглянемо далі найпростіший закон логічного управління динамічним об'єктом.

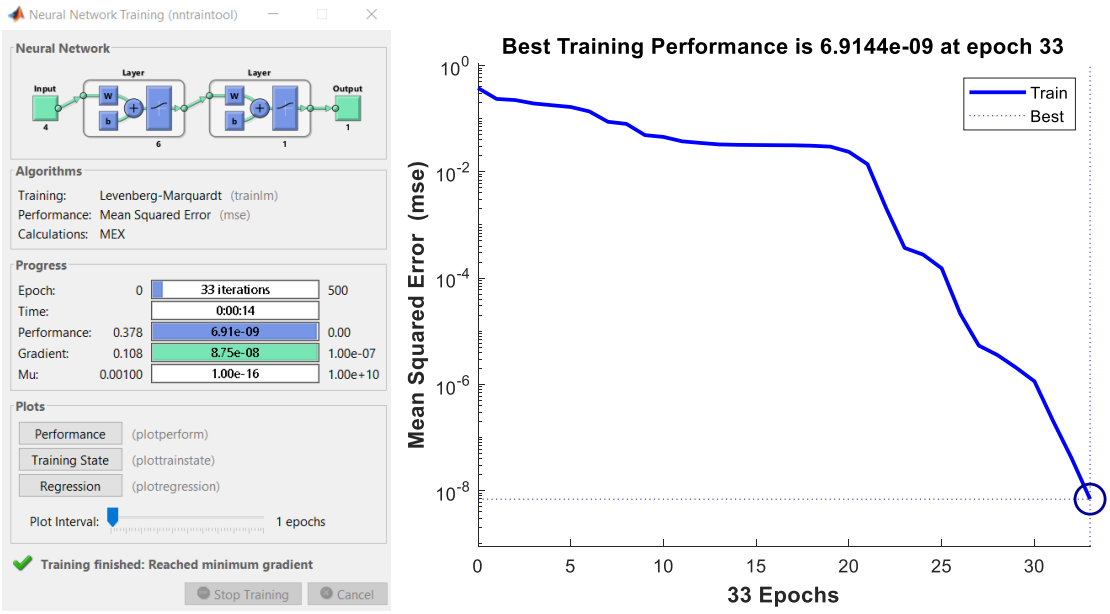


Рисунок 4.6 - Результати навчання НМ класифікації бінарних векторів

Приклад 4.4. Потрібно, щоб нейромережевий регулятор, який отримує на вході помилку управління e, реалізовував релейний закон управління:

$$u = \begin{cases} 1, & e \geq 0, \\ -1, & e < 0. \end{cases}$$

MatLab програма:

```
w=2*pi;
for i=1:300 % Формування тренувального відбору проб
time(i)=0.01*i;
e(i)=(exp(-time(i)))*sin(w*time(i));
if (e(i)>0.0) t(i)=1;
```

```
elseif (e(i)<0.0) t(i)=-1;
end
end
plot(time,e);  grid on;  hold on;
net = newff([min(e) max(e)],[10 1],{'tansig','purelin'});
net.trainParam.epochs = 1000;
net = train(net,e,t);
u=sim(net,e); plot(time,u,'+'); legend('помилка','управління');
xlabel('t')
```

Роботу релейного регулятора ілюструє рисунок 4.7, результати навчання НМ наведені на рисунку 4.8.

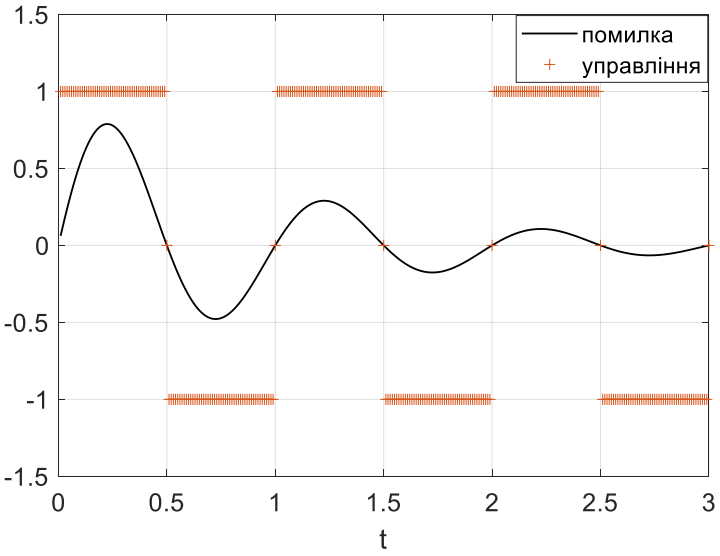


Рисунок 4.7 - Нейромережевий релейній регулятор

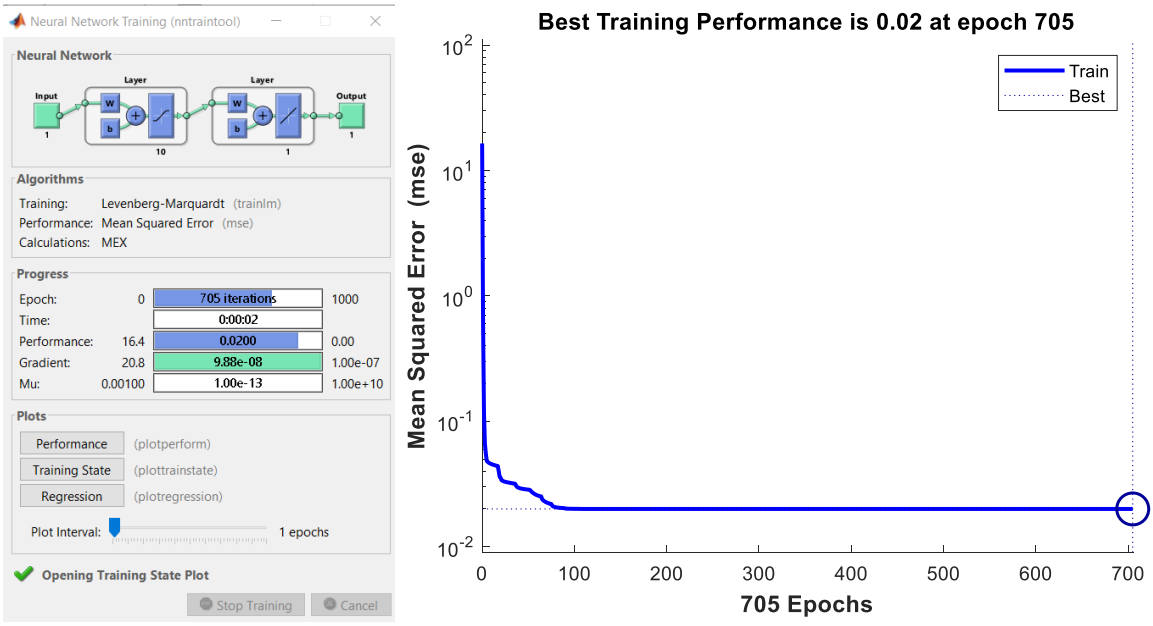


Рисунок 4.8 - Результати навчання НМ

4.4 Апроксимація функцій

У задачах апроксимації функцій потрібно відновити функціональну залежність на підставі обмеженого набору відомих точок. Цю проблему ілюструють такі приклади.

Приклад 4.5. Апроксимація функцій. MatLab програма:

```
P = [0 1 2 3 4 5 6 7 8 9 10];
T = [0 1 2 3 4 3 2 1 2 3 4];
net = newff([0 10],[5 1],{'tansig', 'purelin'});% функція активації
% першого шару - Гіперболічний тангенс tansig
% другого шару - Лінійна purelin
net.trainParam.goal=0.01;
net.trainParam.epochs = 50;
net = train(net,P,T);
X = linspace(0,10);
Y = sim(net,X);
figure(1);
plot(P,T,'ko',X,Y);grid on;  hold on;
```

Результати апроксимація функції, заданої набором точок наведені на рис. 4.9.

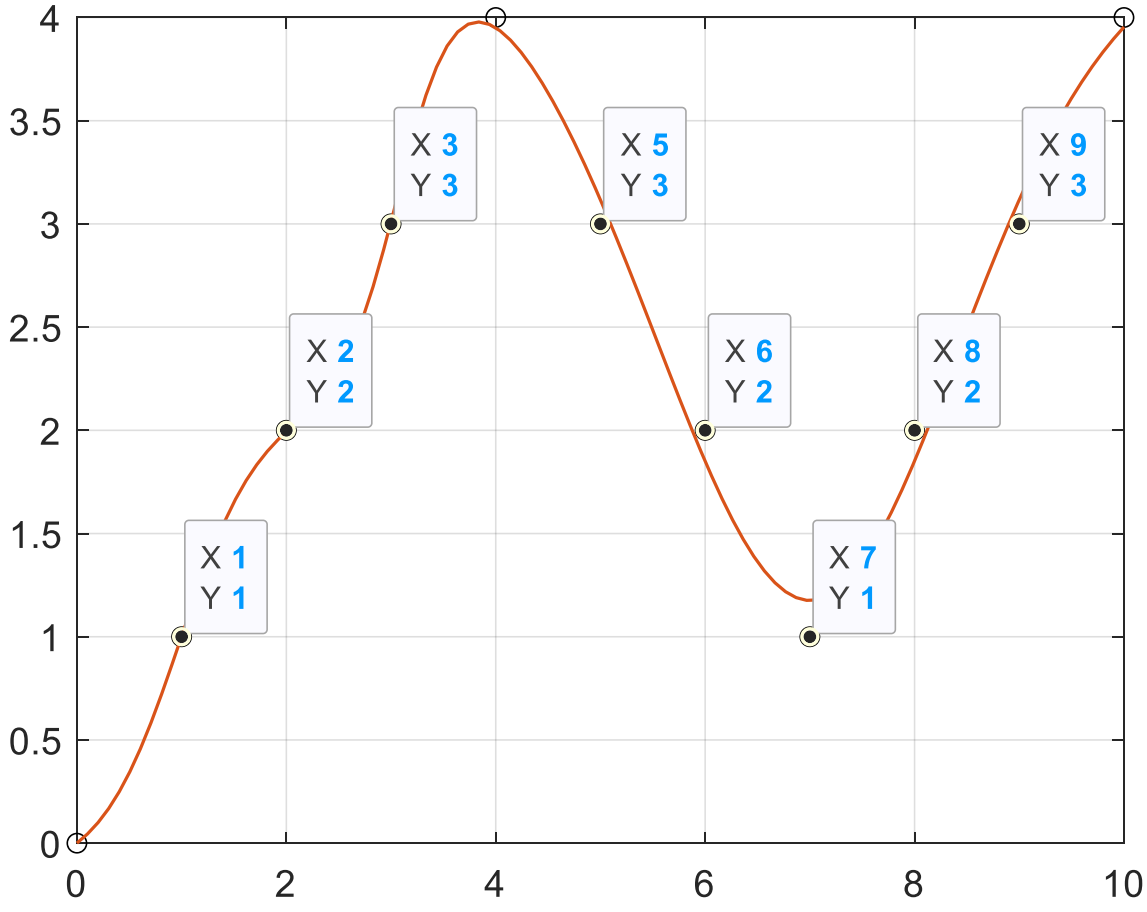


Рисунок 4.9 - Апроксимація функції, заданої набором точок

У цьому прикладі вектори *P* і *T* задають 11 опорних точок, якими потрібно побудувати безперервну функцію. За результатами моделювання наведеними на рисунку 4.10 нейронна мережа має два шари, прихований шар містить п'ять нейронів. Задано 50 епох навчання за необхідної помилки 0,01. З рисунку 4.9 слід, що завдання вирішено досить якісно.

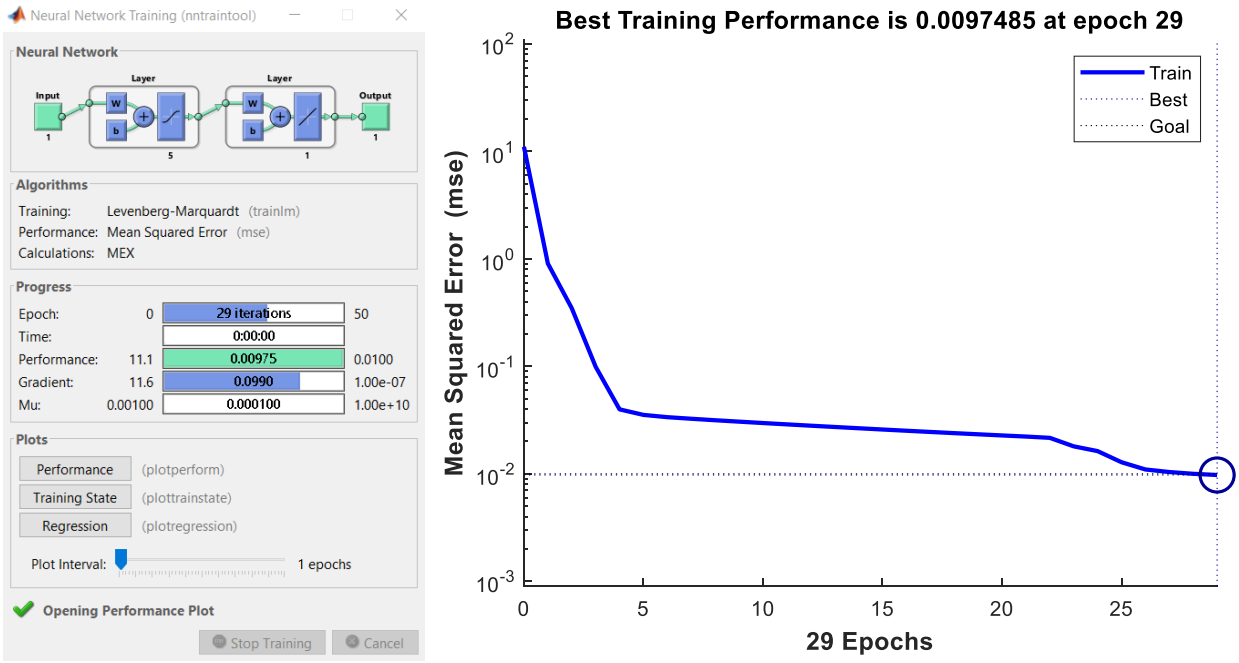


Рисунок 4.10 – Результати математичного моделювання НМ з апроксимація функції, заданої набором точок

Приклад 4.6. Нехай є великий масив точок:

```
>> x = 0:.05:2;
>> y = 1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;
```

Потрібно побудувати ШНМ ПП для апроксимації цього масиву:

```
>> net=newff([0 2], [15,1], {'tansig','purelin'},'trainlm');
>> net.trainParam.show = 50;
>> net.trainParam.epochs =100;
>> net.trainParam.goal = 0.001;
>> P=x;
>> T=y;
>> net1 = train(net, P, T);
```

Результати навчання наведені на рисунку 4.11. Завдана помилка виявилася досягнута на 100-й ітерації.

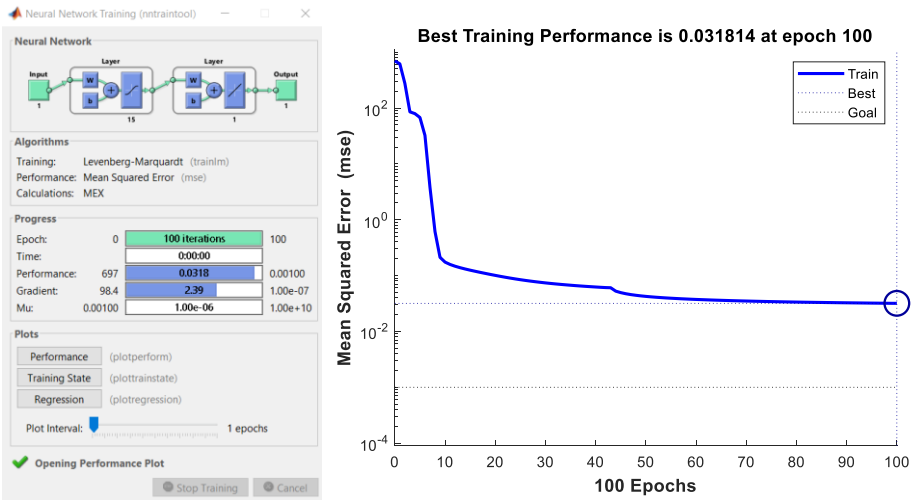


Рисунок 4.11 - Зміна функції помилки у процесі навчання НМ

Примітка: у цьому прикладі використовувався алгоритм навчання trainlm**.**

[net, tr] = trainlm (net, Pd, Tl, Ai, Q, TS, VV) – дана функція повертає ваги та усунення нейронної мережі, використовуючи алгоритм оптимізації Левенберга-Марквардта.

Перевіримо отриманий результат шляхом побудови графіків завданої функції та побудованою НМ:

```
>> A = sim(net1,P);
>> plot(x,y,P,A) ;
>> xlabel(X, P');
>> ylabel('Y, A')
```

Графіки функцій наведенні на рисунку 4.12. За результатами аналізу моделювання можливо побачити, що модельний та завданий графіки повністю збігаються.

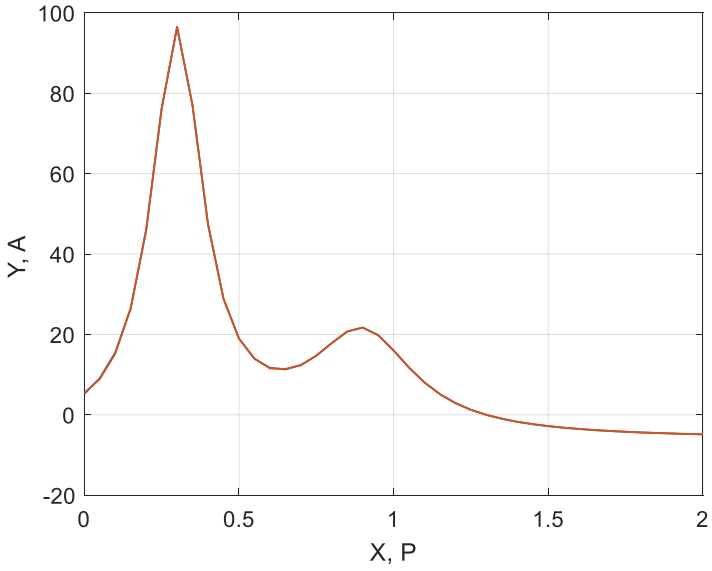


Рисунок 4.12 - Порівняння функції та її апроксимації

Приклад 4.7. Розглянемо апроксимацію функції двох змінних:

$$z=\sin(x)\cos(y).$$

За допомогою стандартних функцій MatLab отримуємо поверхню, наведену на рисунку 4.13:

```
>> x = -2:0.25:2; y = -2:0.25:2;
>> z = cos(x)'*sin(y);
>> figure(1);
>> mesh(x,y,z)
>> xlabel('X');
>> ylabel('Y');
>> zlabel('Z')
```

Опишемо навчальну множину для нейронної мережі:

```
>> P = [x;y];
>> T = z;
```

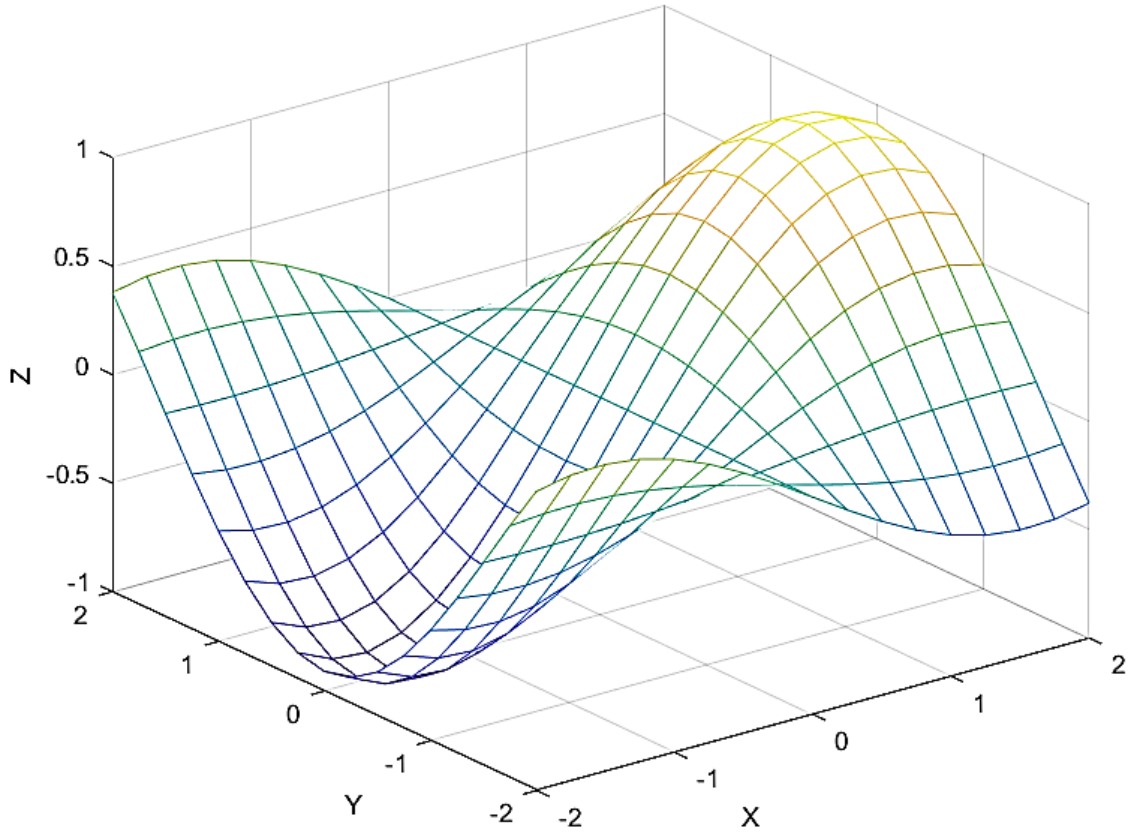


Рисунок 4.13 - Вихідна функція z=sin(*x*)cos(*y*)

При такому способі формування навчальної вибірки кожній парі {*x*; *y*} відповідає стовпець цільової множини. Тому нейронна мережа повинна мати два входи та 17 нейронів на виході:

```
>> net=newff([-2 2; -2 2], [25 17], {'tansig' 'purelin'},'trainlm');
```

```
>> net.trainParam.show = 50;
>> net.trainParam.lr = 0.05;
>> net.trainParam.epochs = 300;
>> net.trainParam.goal = 0.001;
>> net1 = train(net, P, T);
```

Виконаємо перевірку. Побудуємо апроксимаційну поверхню завданої функції (див. рис. 4.14):

```
>> A=sim(net1,P)
>> figure(2);
>> mesh(x,y,A)
>> xlabel('X');
>> ylabel('Y');
>> zlabel('A')
```

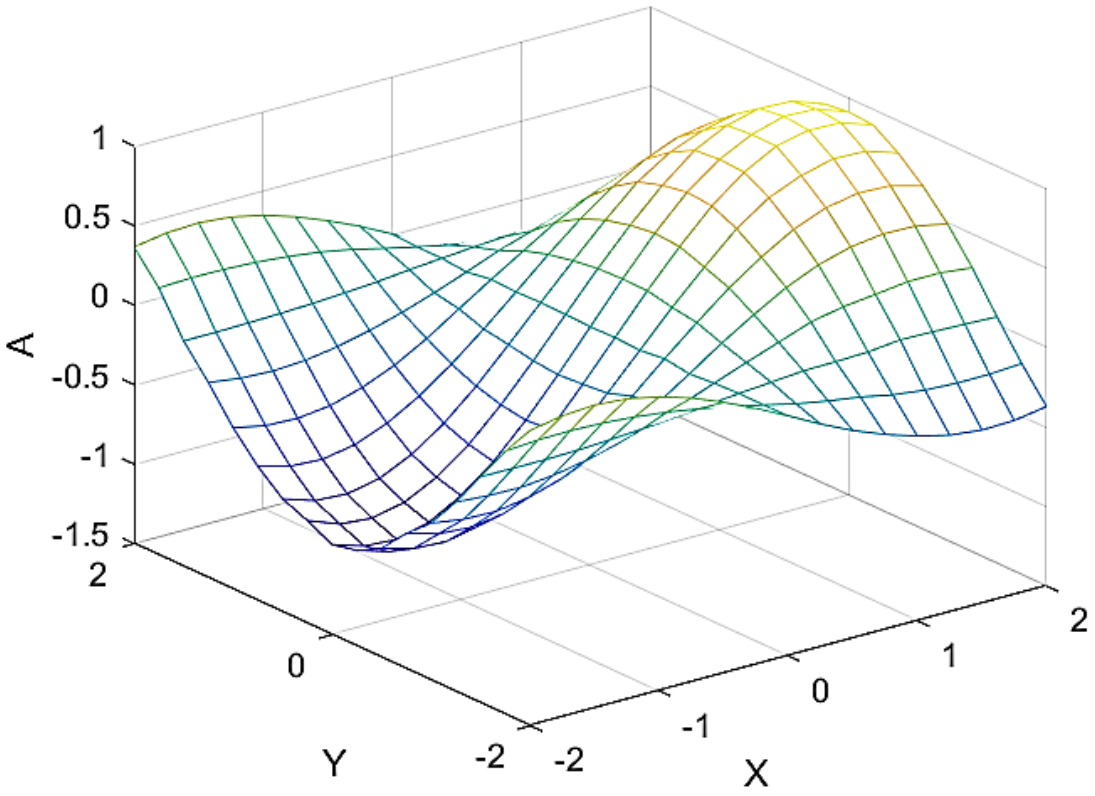


Рисунок 4.14 - Вихід нейронної мережі після навчання

Порівняння результатів на рисунках 4.13 та 4.14 дозволяє говорити про досить високу якість апроксимації.

Приклад 4.8. Апроксимація функції трьох змінних.

Нехай є модель, за допомогою якої можна генерувати вхід-вихідні залежності системи

$$y = 3a + 4ab + 2c + f,$$

де *a*, *b*, *c* - вхідні змінні; *f* – сигнал шуму. Опишемо ці величини як масиви випадкових чисел:

```
>> a = rand(1,100);
>> b = rand(1,100);
>> c = rand(1,100);
>> f = rand(1,100)*0.025;
>> y = 3*a + 4*a.*b + 2*c + f;
```

Навчальна множина задається у вигляді

```
>> P = [a; b; c];
>> T = y;
```

Опишемо ШНМ та запустимо процес навчання:

```
>> net = newff([0 1; 0 1; 0 1], [4 1], {'tansig', 'purelin'});
>> net = train(net, P,T)
```

Для оцінки якості роботи ШНМ виконаємо команди:

```
>> out=sim(net,P);
>> t=1:100;
>> plot(t,y-out)
```

Результат наведено на рисунку 4.15.

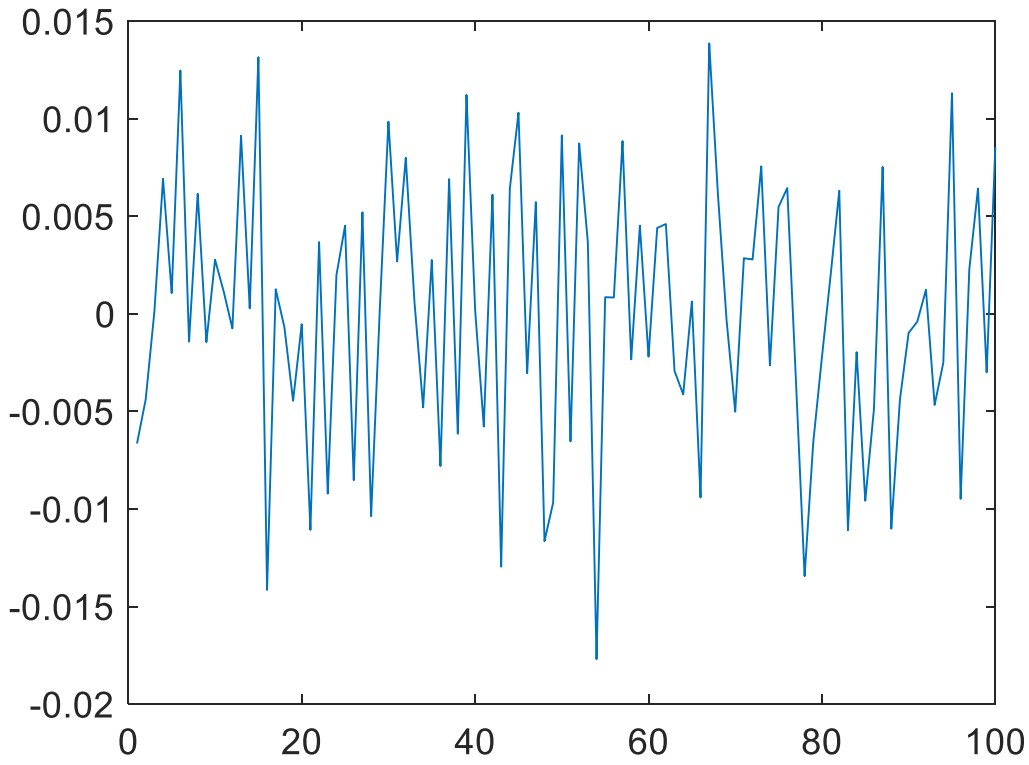


Рисунок 4.15 - Помилка виходу нейронної мережі

При нейромережевій апроксимації функції за точками може виникати явище перенавчання. У наступному прикладі ШНМ після навчання відтворює вплив шумів, накладених на навчальну вибірку.

Приклад 4.9. Явище перенавчання:

MatLab програма:

```
>> net = newff([-1 1],[30,1],{'tansig','purelin'});
% двошарова ШНМ ПП
>> net.trainParam.epochs = 500;
>> p = [-1:0.05:1];        % область визначення функції
>> t1 = sin(2*pi*p);       % еталона функція
>> t = sin(2*pi*p)+0.1*randn(size(p));
% функція з шумом
>> [net,tr] = train(net,p,t);
>> t2 = sim(net,p);
>> figure(1); plot(p,t,'+',p,t2,'-',p,t1,':')
>> legend('вхід','вихід','еталон'); grid on
```

Результат подано на рисунку 4.16.

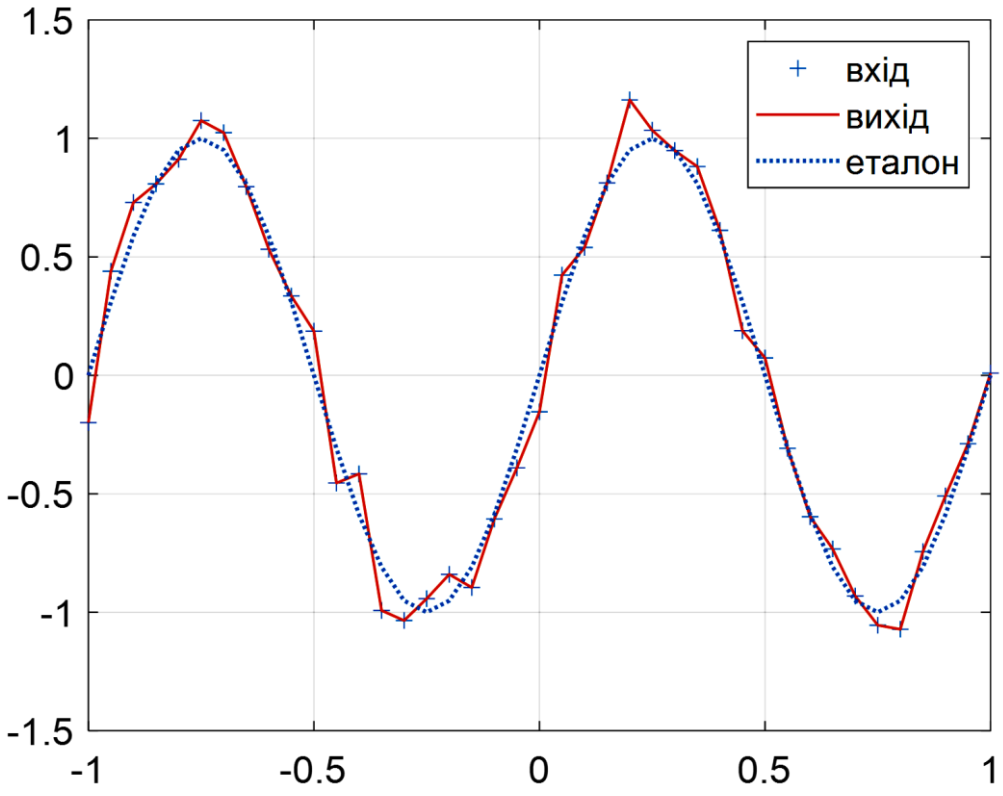


Рисунок 4.16 - Приклад перенавчання нейронної мережі

Вихід навченої мережі відповідає всім точкам зашумленої навчальної вибірки, місцями значно відступаючи від еталонного процесу.

Для усунення ефекту перенавчання можна використовувати розширений варіант команди train, в якому, крім навчальної вибірки, вказується також перевірна вибірка:

```
>> val.P = [-0.975: 0.05: 0.975];
% формування перевірочної вибірки
>> val.T = sin(2*pi*val.P) + 0.1*randn(size(val.P));
>> [net, tr] = train(net, p, t, [], [], val);
```

Результат навчання цього варіанта команди представлений на рис. 4.17.

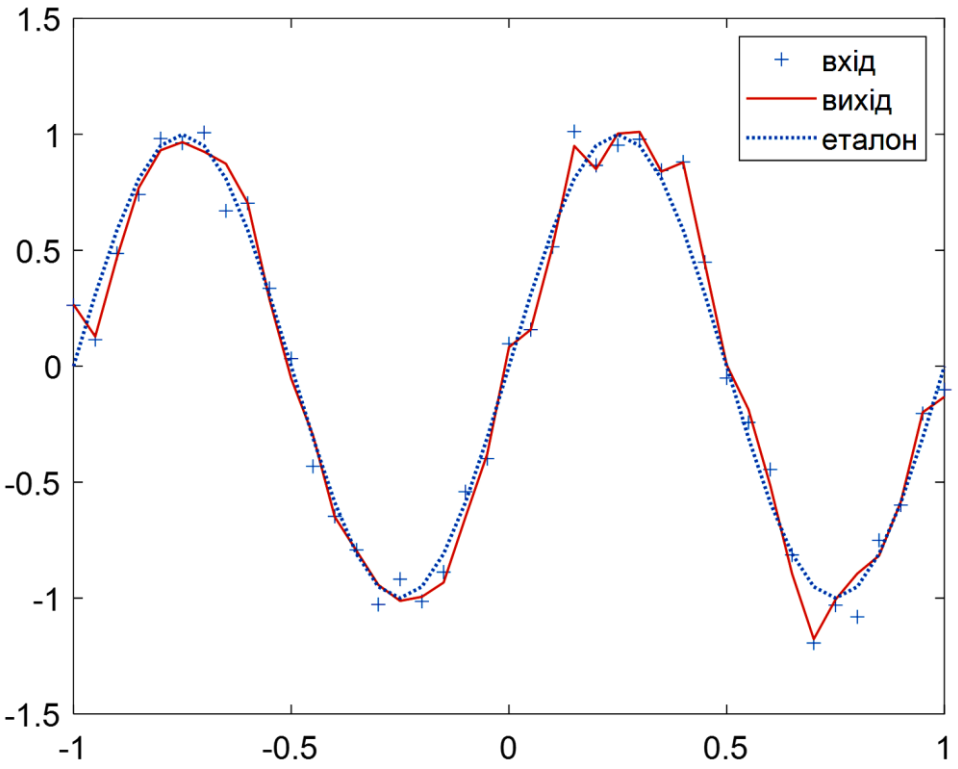


Рисунок 4.17 - Контроль перенавчання нейронної мережі

Навчання тут припиняється раніше, коли контрольна помилка починає значно перевищувати помилку навчання. Завдяки цьому вихід нейронної мережі не реагує на окремі аномальні точки навчальної множини (див. рис. 4.16).

4.5 Математичне моделювання статичних залежностей з використанням ШНМ ПП

Важливою є можливість опису залежності вхід-вихід динамічних об'єктів за допомогою ШНМ ПР.

Приклад 4.10. Розглянемо моделювання за допомогою ШНМ ПП динамічної системи, що описується функцією Бесселя

$$t^2\frac{d^2y}{dt^2} + t\frac{dy}{dt} + (t^2 - a^2)y = 0.$$

Графік функції Бесселя представлено на рис. 4.18. Програма MatLab графіка:

```
t=0:0.1:20;
y=besselj(1,t);
plot(t,y)
grid
xlabel('t,c')
ylabel('y(t)')
```

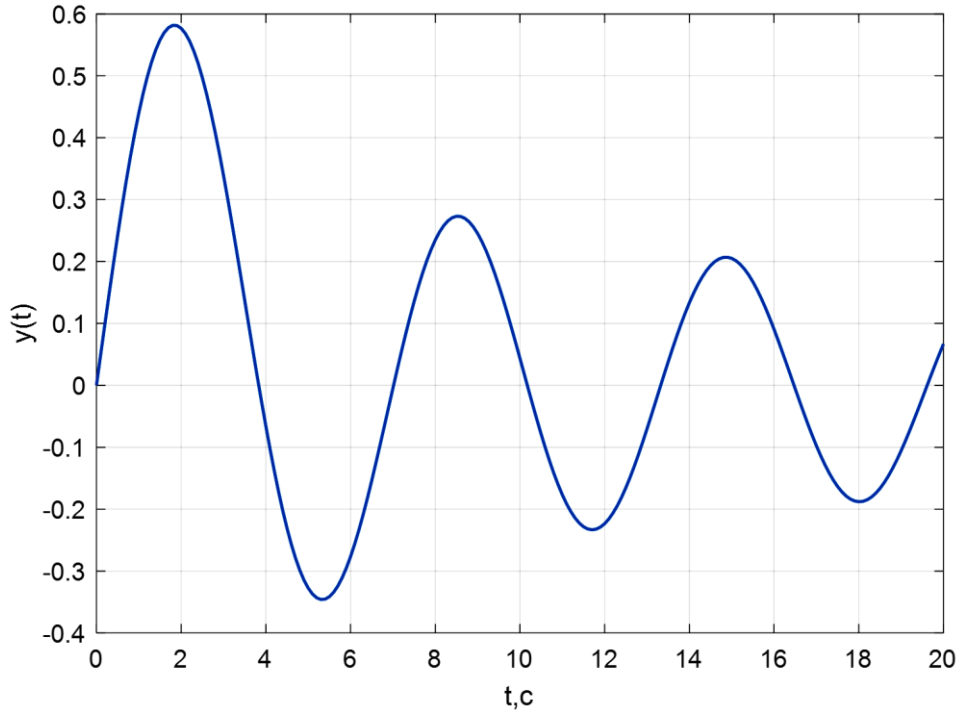


Рисунок 4.18 -. Функція Бесселя

Для моделювання будемо використовувати двошарову мережу з десятьма нейронами прихованого шару:

```
>> net=newff([0 20], [10,1], {'tansig','purelin'},'trainlm');
>> P=t; T=y;
>> net.trainParam.show = 50;
>> net.trainParam.lr = 0.05;
>> net.trainParam.epochs = 300;
>> net.trainParam.goal = 0.001;
>> net1 = train(net, P, T);
```

Для навчання знадобилося лише 7 епох (рис. 4.19).

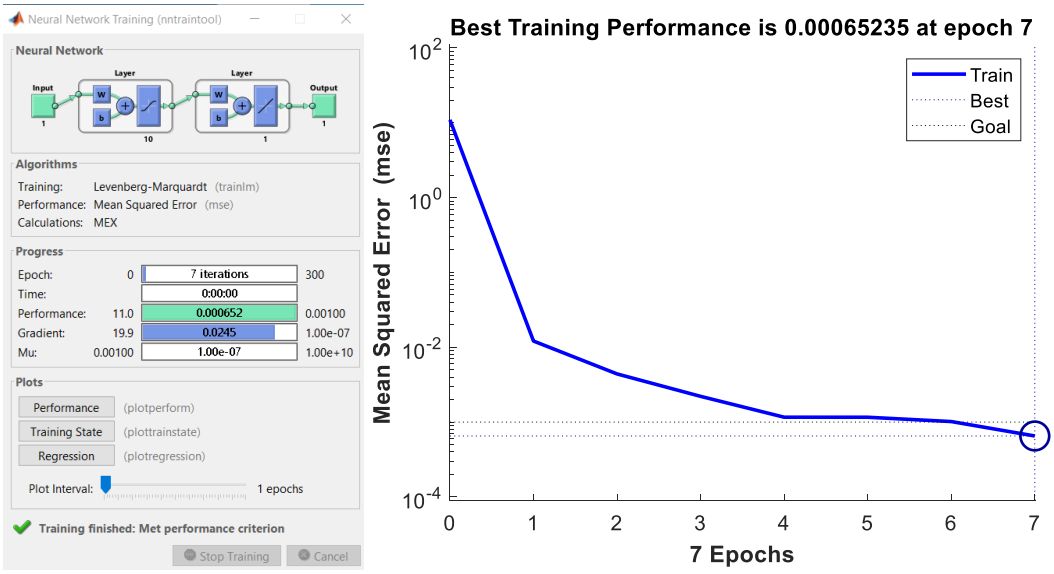


Рисунок 4.19 - Зміна помилки у процесі навчання

Виконаємо моделювання роботи мережі:

```
>> A= sim(net1,P);
```

Для перевірки якості роботи мережі розглянемо помилку моделювання (див. рис. 4.20):

```
>> figure(1); plot(P,T-A)
```

Як випливає з рисунку 4.20, в кінці перехідного процесу помилка неприпустимо зростає, тому параметри нейронної мережі потребують корекції. На рис. 4.21 Наведено графіки порівняння завданої функціїї з фнкцією побудованою за допомогою НМ

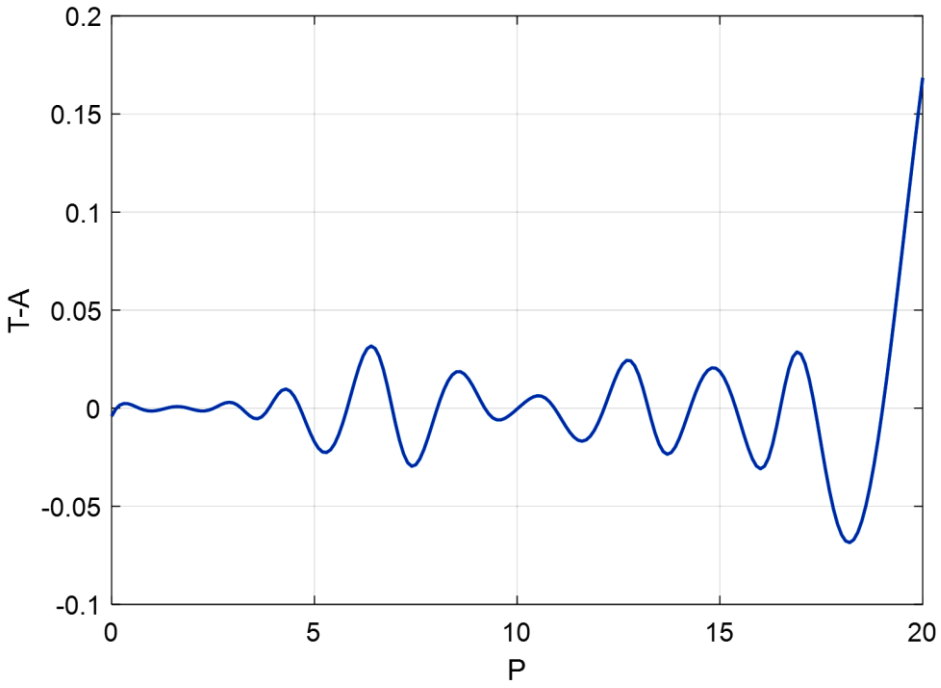


Рисунок 4.20 - Помилка на різних ділянках перехідного процесу

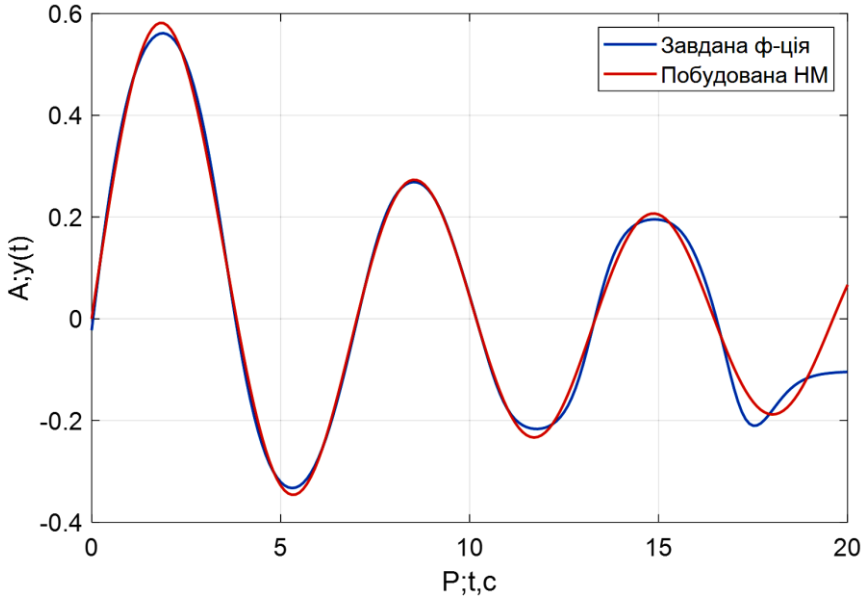


Рисунок 4.21 – Порівняння завданої функціїї з фнкцією побудованою за допомогою НМ

Приклад 4.11. Нехай є лінійна динамічна ланка 2-го порядку. Його можна описати у вигляді диференціального рівняння або у вигляді відповідної передавальної функції

$$\frac{d^2y(t)}{dt^2} + 0{,}5\frac{dy(t)}{dt} + y(t) = x(t) \Leftrightarrow W(s) = \frac{Y(s)}{X(s)} = \frac{1}{s^2 + 0.5s + 1}.$$

На рисунку 4.22 представлена зібрана в MatLab Simulink схема для отримання реакції ланки на одиничний стрибок.

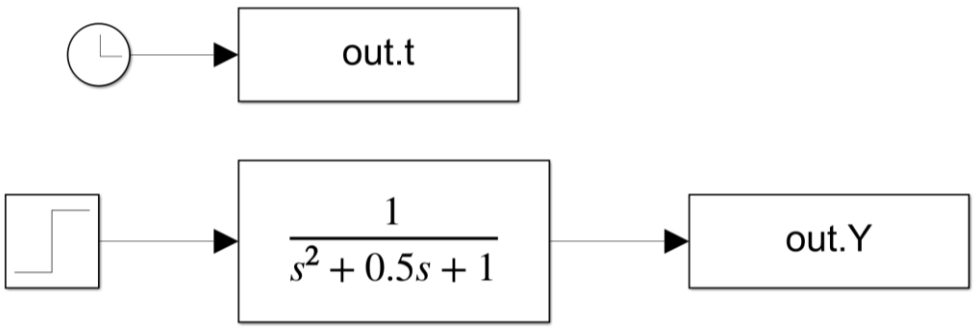


Рисунок 4.22 - Модель системи в MatLab Simulink лінійної динамічної ланки 2-го порядку

Виконаємо моделювання (примітка: PR11 - ім'я файлу моделі, 20 – час моделювання):

```
>> simOut=sim('PR11',20);
```

та побудуємо графік перехідного процесу (див. рис. 4.23):

```
>> figure(1);
>> plot(simOut.t,simOut.Y);
>> xlabel('t,c');
>> ylabel('Y(t)');
>> grid;
```

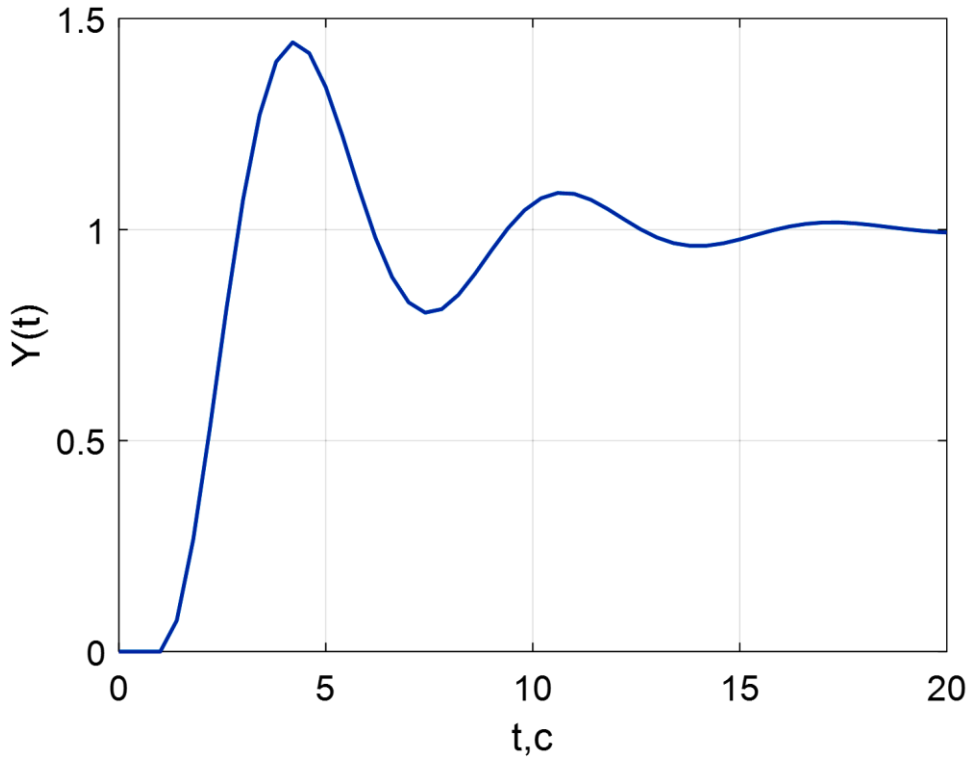


Рисунок 4.23 - Вихідний сигнал динамічного об’єкта

Задаємо навчальну вибірку, параметри нейронної мережі та навчання згідно програми MatLab:

```
P = simOut.t';
T = simOut.Y';
net=newff([0 20], [15,1], {'tansig','purelin'},'trainlm');
net.trainParam.show = 50;
net.trainParam.lr = 0.005;
net.trainParam.epochs = 100;
net.trainParam.goal = 0.0001;
net1 = train(net, P, T); % виконуємо моделювання
A= sim(net1,P);
figure(2); plot(P,A)
grid;
```

Потім експортуємо отриману нейронну мережу у Simulink (де 0,01 - постійний крок інтегрування за часом):

>> gensim(net1,0.01)

За результатом виконання команди gensim формується блок Custom Neural Network (див. рис 4.24).

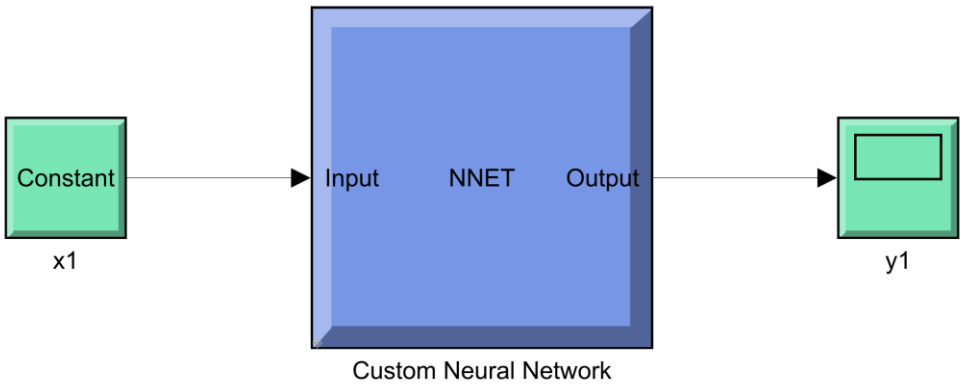


Рисунок 4.24 - Блок Custom Neural Network синтезованої НМ

Сформований блок додаєм до математичної моделі лінійної динамічної ланки 2-го порядку, як зображено на рисунку 4.25.

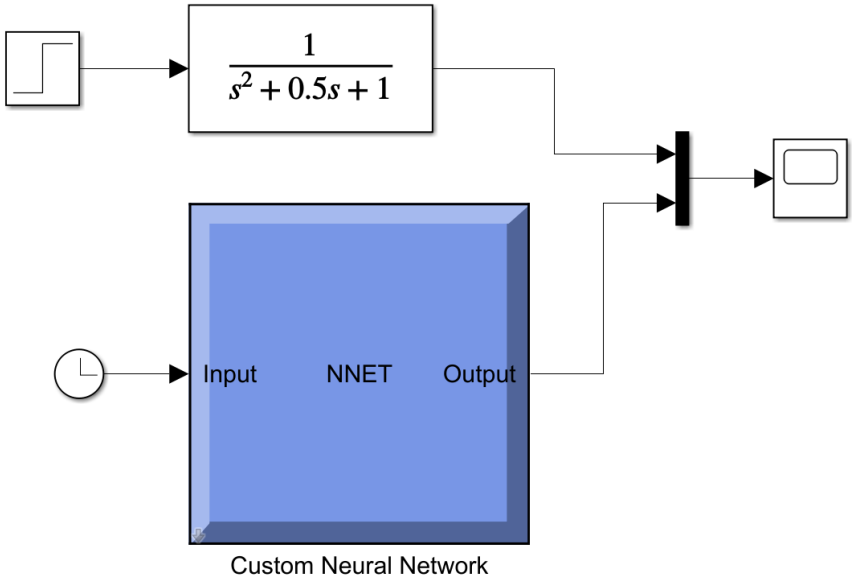


Рисунок 4.25 -. Порівняння динамічної ланки та його апроксимації

Схема наведена на рисунку 4.24, дозволяє порівняти вихідні сигнали передавальної функції та нейронної мережі. З графіків порівняння сигналів зображених на рисунку 4.26 можливо зробити висновок, що вихідні сигнали досить близькі.

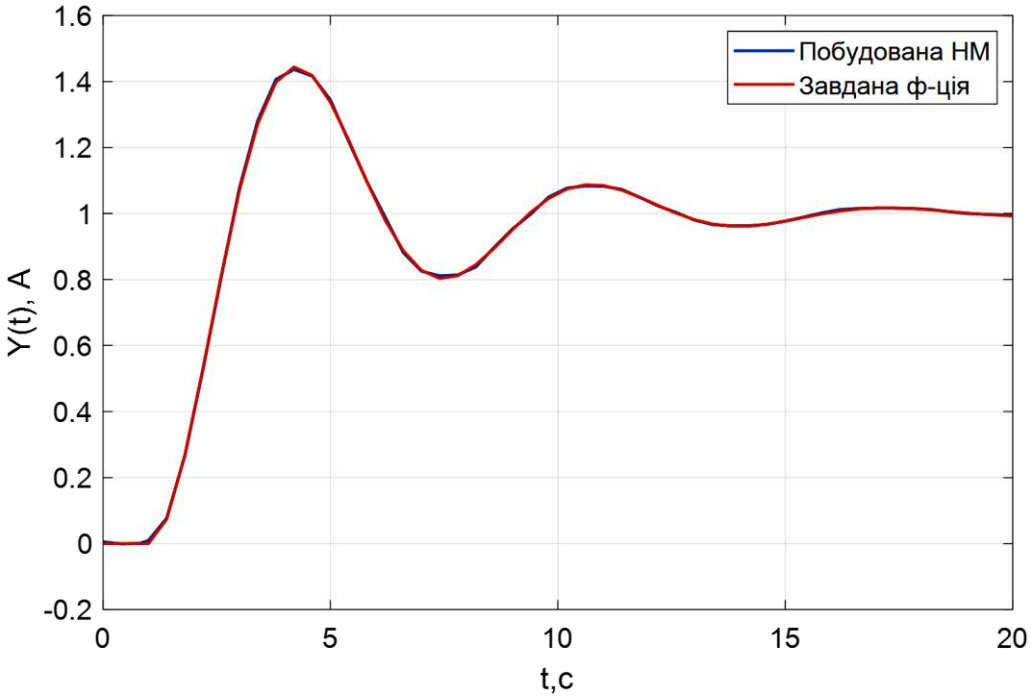


Рисунок 4.26 - Виходи динамічної ланки та нейронної мережі

Однак очевидно, що при зміні вхідного сигналу динамічної системи (наприклад, при заміні step на sine wave) вихідний сигнал також буде змінюватися, і буде потрібно формування нової навчальної вибірки (рис. 4.27).

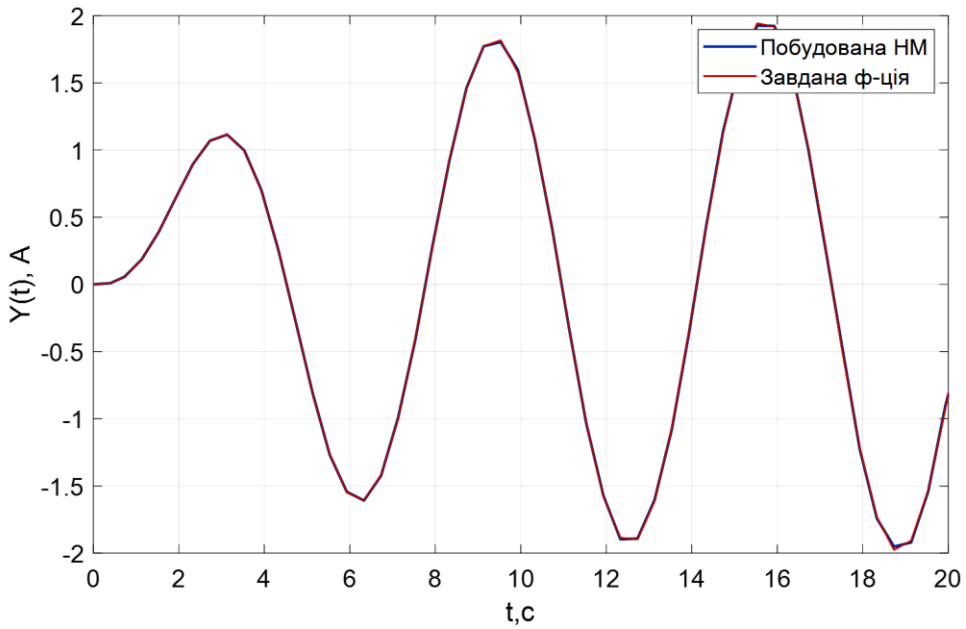


Рисунок 4.27 - Порівняння реакцій на синусоїдальний сигнал після перенавчання НМ

Таким чином, при зміні вхідного сигналу потрібне перенавчання нейронної мережі. У цьому полягає статичність її поведінки.

4.6 Масштабування та відновлення даних

Працюючи з ШНМ ПП часто виникає необхідність передобробки та постобробки інформації. У MatLab передбачено набір команд для виконання цих операцій.

-10	0
-7,5	7,07
-5	-10
-2,5	-7,07
0	0
2,5	7,07
5	10
7,5	7,07
10	0

Дані зчитуються командою

```
>> load data.txt;
>> P=data(1:9,1);
>> T=data(1:9,2);
```

Наступна команда виконує нормалізацію навчальних даних:

```
>> [pn,minp,maxp,tn,mint,maxt] = premnmx(P',T')

    pn =
-1.0000 -0.7500 -0.5000 -0.2500 0 0.2500 0.5000 0.7500 1.0000
    minp = -10
    maxp =      10
    tn =
0 0.7070 -1.0000 -0.7070 0 0.7070 1.0000 0.7070 0
    mint = -10
    maxt =      10
```

Приклад 4.12. Спочатку за допомогою функції premnmx виконаємо масштабування навчальної послідовності до діапазону [-1 1], потім створимо та навчимо ШНМ ПП, виконаємо її моделювання та відновлення виходу за допомогою функції postmnmx:

```
>> P = [-0.92 0.73 -0.47 0.74 0.29; -0.08 0.86 -0.67 -0.52 0.93];
>> t = [-0.08 3.40 -0.82 0.69 3.10];
>> [pn,minp,maxp,tn,mint,maxt] = premnmx(P,t);
>> net = newff(minmax(pn),[5 1],{'tansig' 'purelin'},'trainlm');
>> net = train(net,pn,tn); grid on
>> an = sim(net,pn)
```

```
        an = -0.6493 1.0000 -1.0000 -0.2844 0.8578
>> a = postmnmx(an,mint,maxt)

        a = -0.0800 3.4000 -0.8200 0.6900 3.1000
```

Аналогічно використовується функція prestd, яка виконує масштабування навчальної послідовності за законом розподілу з параметрами [0 1], а відновлення виходу відбувається за допомогою функції poststd.

5 РАДІАЛЬНІ НЕЙРОНІ МЕРЕЖІ

5.1 Структура радіальної нейронної мережі

Радіальні нейронні мережі (Radial Basis Function – RBF), це двошарові мережі (без урахування розподільчого вхідного шару) прямого розповсюдження (див. рис. 5.1).

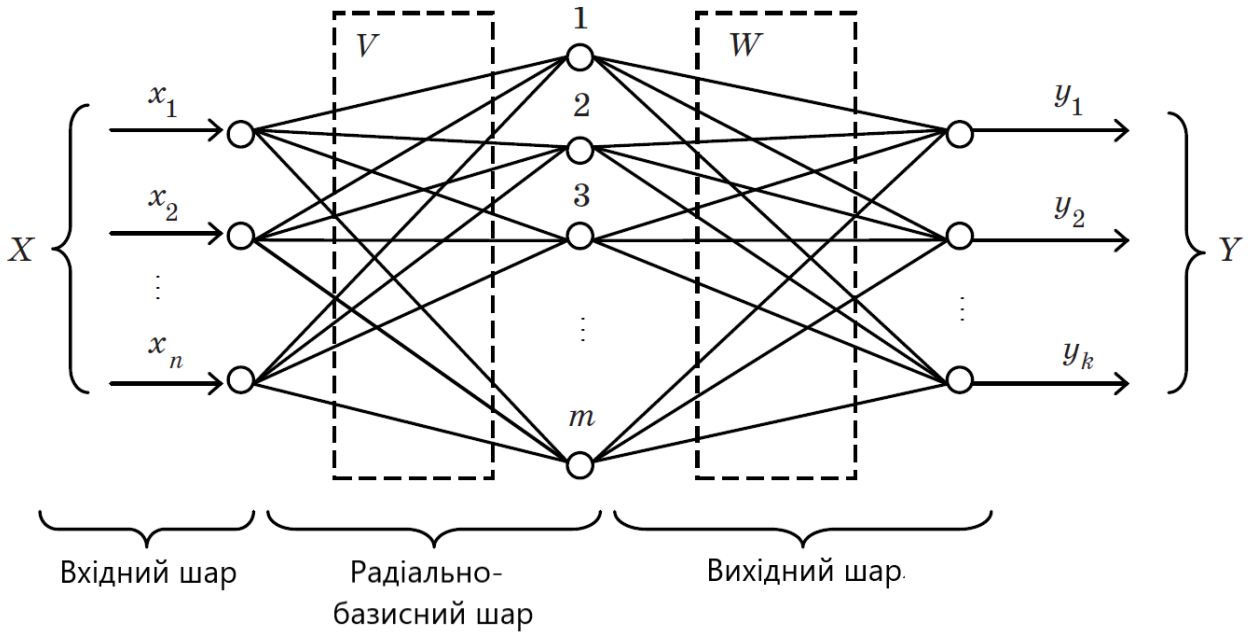


Рисунок 5.1 - Структура RBF-мережі зі скалярним виходом

Радіальна нейронна мережа подібно до багатошарової мережі прямого поширення є універсальним апроксиматором.

Радіально-базисні функції – спеціальний клас функцій, значення яких монотонно зменшується (збільшується) із збільшенням відстані від центру.

Всі ваги радіально-базового шару вважаються рівними одиниці, і роботу *i*-го нейрона RBF-шару можна описати формулою

$$f_i(X) = \varphi(\|X - C_i\|),$$

де C_i – вектор центру активаційної RBF функції нейрона: $X, C \in R^n$. Таким чином, вхідний вектор та вектор центру мають однакову розмірність.

Як радіальна базисна функція φ зазвичай використовується гауссова функція

$$\varphi(\|X - C_i\|) = exp\left(\frac{\|X - C_i\|^2}{2\sigma_i^2}\right),$$

де σ - ширина «вікна» активації функції.

Як метрика зазвичай використовується евклідова відстань

$$\|X - C_i\| = \sqrt{(x_1 - c_{i1})^2 + (x_2 - c_{i2})^2 + \cdots + (x_n - c_{in})^2}.$$

Таким чином, *i*-й нейрон прихованого шару визначає схожість між вектором вхідним *X* та еталонним вектором C_i.

На рисунку 5.2 наведено гаусову функцію однієї змінної при *c*= 0 і σ= 1.

```
x = -3:0.1:3;
y = gaussmf(x,[1 0]);
plot(x,y)
xlabel('gaussmf, P=[-3 3]')
```

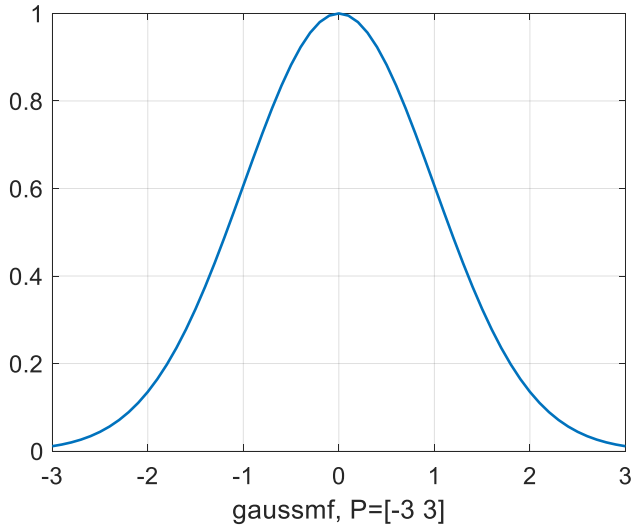


Рисунок. 5.2 - Графік гаусової функції

Як випливає з рис. 5.2, функція активації RBF-нейрону приймає великі значення лише у випадках, коли вхідний образ знаходиться поблизу центру нейрона. Поза областю, «покритою» образами навчальної послідовності, мережа формує лише малі значення на своїх виходах.

Можливі інші варіанти активаційної функції. Наприклад,

$\varphi(\|X - C_i\|) = \|X - C_i\|$ – лінійна функція,

$\varphi(\|X - C_i\|) = \|X - C_i\|^3$ – кубічна функція,

$\varphi(\|X - C_i\|) = (\|X - C_i\|^2 + \sigma^2)^{1/2}$ – мультиквадратична функція.

Головна відмінність RBF-мереж від звичайних багатошарових мереж прямого поширення полягає у функції нейронів прихованого шару. У звичайній багатошаровій мережі кожен нейрон робочого шару реалізує у багатовимірному просторі гіперплощину, а RBF-нейрон – гіперсферу (див. рис. 5.3).

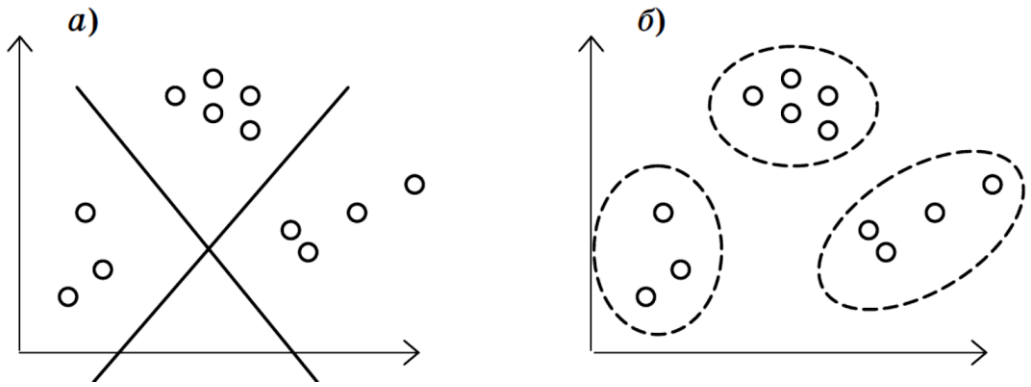


Рисунок 5.3 – Принципи класифікації:
а) з допомогою багатошарового персептрону, б) RBF-мережі

У проблемах, де дані близькі до утворення кругової симетрії, це дозволяє зменшити число нейронів.

Нейрони вихідного шару мають лінійну активаційну функцію. Їхня роль зводиться виключно до зваженого підсумовування сигналів, що генеруються нейронами робочого шару:

$$y_i = \sum_{i=1}^{m} w_{ij} f_i(X), \quad j = \overline{1,k}.$$

Число нейронів вихідного шару визначається характером подання вихідних даних.

5.2 Розрахунок параметрів радіальної нейронної мережі

Розглянемо простий варіант визначення ваг RBF-мережі.

Нехай RBF-мережа має *k* входів та один вихід (див. рис. 5.4).

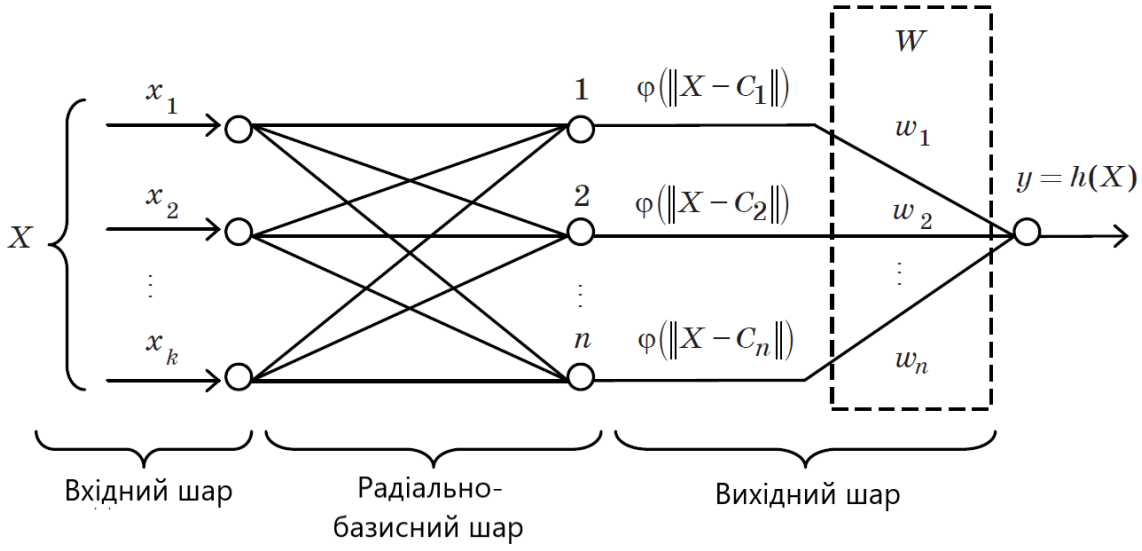


Рисунок 5.4 – Структура RBF-мережі зі скалярним виходом

Виберемо число робочих нейронів $m = n$, де n – кількість навчальних пар, заданих набором $\{(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)\}$, де $X_i = [x_{i1}, x_{i2}, \ldots, x_{ik}]^T$.

Для того, щоб кожен нейрон реагував на «свій» вектор з навчального набору, вважаємо

$$C_i = X_i$$

Вікна активаційної функції σ вибирають досить великими, але так, щоб вони не перекривалися у просторі вхідних сигналів.

Потрібно знайти такі вагові коефіцієнти W, щоб для кожного вхідного вектора з навчального набору виконувалося

$$h(X_i) = y_i$$

Для першого вхідного вектора з навчального набору можна записати

$$h(X_1) = \sum_{i=1}^{m} w_i f_i(X_1) = w_1 f_1(X_1) + w_2 f_2(X_2) + \cdots + w_n f_n(X_n).$$

Для всіх n вхідних векторів при правильному виборі W слід виконувати

$$\begin{vmatrix} f_1(X_1) & f_2(X_1) & \cdots & f_n(X_1) \\ f_1(X_2) & f_2(X_2) & \cdots & f_n(X_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(X_n) & f_2(X_n) & \cdots & f_n(X_n) \end{vmatrix} \begin{vmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n1} \end{vmatrix} = \begin{vmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{vmatrix}.$$

Вводимо позначення для матриць:

$$FW = Y.$$

Тоді

$$W = F^{-1}Y. \qquad (5.1)$$

Остання формула дозволяє розрахувати ваги RBF-мережі з одним виходом при числі нейронів прихованого шару, що дорівнює кількості навчальних пар.

Далі буде показано, що аналогічний результат може бути отриманий при довільному числі нейронів вихідного шару RBF-мережі, якщо кількість навчальних пар дорівнює кількості нейронів прихованого шару.

Нехай вихідний шар містить p нейронів, тому вектор виходу має вигляд

$$Y_i = [y_{i1}, y_{i2}, \cdots, y_{in}]^T.$$

Визначимо ваги нейронів вихідного шару $w_{ij},\ j = \overline{1,n},\ j = \overline{1,p}$.

Для цього мережі надається весь набір шаблонів, так що для всіх n вхідних векторів можна записати

$$\begin{vmatrix} f_1(X_1) & f_2(X_1) & \cdots & f_n(X_1) \\ f_1(X_2) & f_2(X_2) & \cdots & f_n(X_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(X_n) & f_2(X_n) & \cdots & f_n(X_n) \end{vmatrix} \begin{vmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{1p} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{pp} \end{vmatrix} = \begin{vmatrix} y_{11} & y_{12} & \cdots & y_{1p} \\ y_{21} & y_{22} & \cdots & y_{1p} \\ \vdots & \vdots & \vdots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{pp} \end{vmatrix}.$$

Рядки матриці *F* відповідають виходам нейронів прихованого шару кожного вхідного шаблону. Стовпці матриці *W* відповідають ваговим коефіцієнтам нейронів вихідного шару. Рядки матриці *Y* описують виходи нейронів другого (вихідного) шару кожного вхідного вектора.

Матриця ваг *W* може бути розрахована згідно (5.1).

Таким чином, ваги RBF-мережі можуть бути розраховані за тренувальними шаблонами. Якщо навчальні пари вибрані вдало, то мережа успішно виконуватиме інтерполяцію і породжуватиме близькі вихідні сигнали для близьких вхідних сигналів.

Однак у практичних завданнях умова *m* = *n* зазвичай неприйнятна, оскільки вимагає використання дуже великої кількості нейронів. Крім того, мережа стає надмірно чутливою до шумів у навчальній вибірці. Таким чином, зазвичай *m* << n (кількість нейронів прихованого шару *m* менше числа навчальних пар *n*), і потрібно знайти наближене рішення задачі апроксимації.

Процес підбору наближеного значення ваги може розглядатися як завдання мінімізації цільової функції, що описує помилку виходу мережі.

Для оптимального вибору коефіцієнтів RBF-мережі можна використовувати метод найменших квадратів.

Розглянемо RBF-мережу з одним вихідним та m прихованими нейронами:

$$y_i = \sum_{i=1}^{m} w_i f_i(X). \qquad (5.2)$$

Нехай необхідно апроксимувати залежність, задану безліччю вхід-вихідних даних (навчальна вибірка):

$$\{(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)\},$$

Для якісної апроксимації потрібно мінімізувати помилку виходу мережі, задану формулою

$$E = \sum_{i=1}^{n} (h(X_i) - y_i)^2. \quad (5.3)$$

Розглянемо похідну (5.3)

$$\frac{\partial E}{\partial w_j} = 2 \sum_{i=1}^{n} (h(X_i) - y_i) \frac{\partial h(X_i)}{\partial w_j}. \quad (5.3)$$

Відповідно (5.2)

$$\frac{\partial h(X_i)}{\partial w_j} = f_i(X_i),$$

отже,

$$\frac{\partial E}{\partial w_j} = 2 \sum_{i=1}^{n} (h(X_i) - y_i) f_i(X_i).$$

В точці оптимуму

$$\frac{\partial E}{\partial w_j} = 2 \sum_{i=1}^{n} (h(X_i) - y_i) f_i(X_i) = 0.$$

Або

$$\sum_{i=1}^{n} f_j(X_i) h_i(X_i) = \sum_{i=1}^{n} f_j(X_i) y_i.$$

Позначимо

$$F_j = \begin{vmatrix} f_j(X_1) \\ f_j(X_2) \\ \vdots \\ f_j(n) \end{vmatrix}, H_j = \begin{vmatrix} h_j(X_1) \\ h(X_2) \\ \vdots \\ h_j(n) \end{vmatrix}, Y = \begin{vmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{vmatrix}.$$

Тоді

$$F_j^T H = F_j^T Y, \quad j = \overline{1,n}$$

$$\begin{vmatrix} F_1^T H \\ F_2^T H \\ \vdots \\ F_n^T H \end{vmatrix} = \begin{vmatrix} F_1^T Y \\ F_2^T Y \\ \vdots \\ F_n^T Y \end{vmatrix},$$

$$F^T H = F^T Y$$

де $F = [F_1 \;\; F_2 \;\; \cdots \;\; F_n]$

Оскільки

$$H = \begin{vmatrix} h(X_1) \\ h(X_2) \\ \vdots \\ h(X_n) \end{vmatrix} = \begin{vmatrix} \sum_{j=1} w_j f_j(X_1) \\ \sum_{j=1} w_j f_j(X_2) \\ \vdots \\ \sum_{j=1} w_j f_j(X_n) \end{vmatrix} = \begin{vmatrix} f_1(X_1) & f_2(X_1) & \cdots & f_n(X_1) \\ f_1(X_2) & f_2(X_2) & \cdots & f_n(X_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(X_n) & f_2(X_n) & \cdots & f_n(X_n) \end{vmatrix} \begin{vmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n1} \end{vmatrix} = FW,$$

можливо записати

$$F^T FW = F^T Y$$

та остаточно

$$W = (F^T F)^{-1} F^T H = F^+ Y$$

де F^+ - псевдозворотна матриця.

Приклад 5.1. Нехай заданий набір із трьох пар точок (p = 3):

{(0,9; 1), (2,1; 1,9), (3,1; 3)}.

Потрібно апроксимувати цю залежність функцією

$$y(x) = w_1 h_1(x) + w_2 h_2(x)$$

де $h_1(x)$ та $h_2(x)$ – виходи радіально-базових нейронів, заданих у вигляді

$$h_1(x) = \exp(-(x - 1.5)^2), \;\; h_2(x) = \exp(-(x - 2.5)^2)$$

Потрібно знайти невідомі коефіцієнти w_1 та w_2:

$$F = \begin{bmatrix} h_1(x_1) & h_2(x_1) \\ h_1(x_2) & h_2(x_2) \\ h_1(x_3) & h_2(x_3) \end{bmatrix} = \begin{bmatrix} 0{,}6977 & 0{,}0733 \\ 0{,}6977 & 0{,}8521 \\ 0{,}0773 & 0{,}6977 \end{bmatrix}, \;\; Y = \begin{bmatrix} 1 \\ 1{,}9 \\ 3 \end{bmatrix}$$

$$(F^T F) = \begin{bmatrix} 0{,}9795 & 0{,}7024 \\ 0{,}7024 & 1{,}2188 \end{bmatrix},$$

$$(F^T F)^{-1} = \begin{bmatrix} 1{,}7398 & -1{,}0026 \\ -1{,}0026 & 1{,}3982 \end{bmatrix},$$

$$w = (F^T F)^{-1} F^T Y = \begin{bmatrix} 0{,}1244 \\ 3{,}0373 \end{bmatrix}.$$

На рисунку 5.5 наведено результат апроксимації. Отримана якість явно невисока порівняно з лінійною функцією $z(x)$, що забезпечує у разі найкращу апроксимацію вихідних даних.

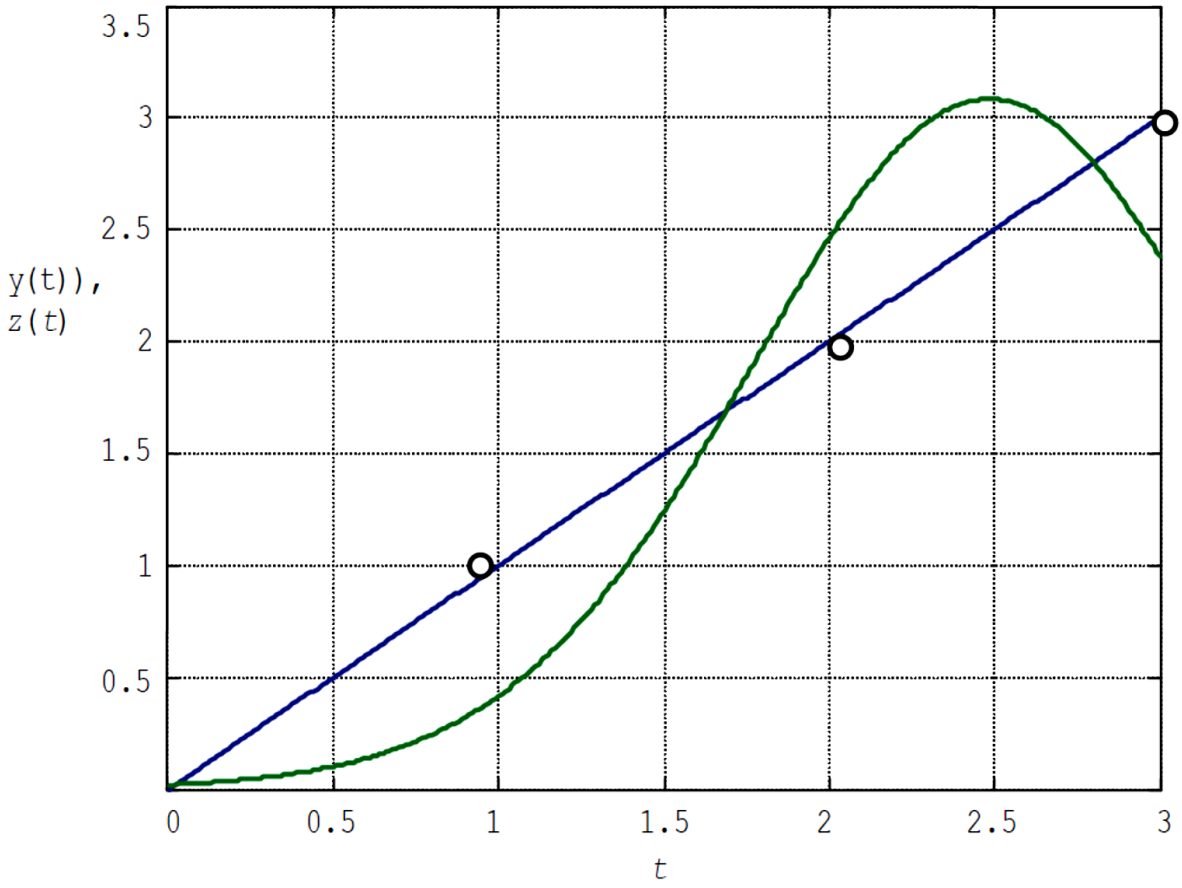


Рисунок 5.5 - Апроксимація функції, заданої трьома точками

Таким чином, якщо параметри гаусової функції (центр та радіус) задані, то завдання знаходження ваги вихідного шару RBF-мережі може бути вирішена методами лінійної алгебри – методом псевдозворотних матриць.

5.3. Навчання радіальної нейронної мережі

У більшості випадків вибір центрів і радіусів активаційних функцій є складним завданням, що вимагає аналізу навчальної вибірки. При цьому можна виділити три взаємозалежні проблеми, пов'язані з вибором:

- числа нейронів прихованого шару;
- центрів та радіусів нейронів;
- ваги вихідного шару.

Для вирішення першого завдання може бути використана, наприклад, вершинна кластеризація [50]. Отримана кількість кластерів приблизно відповідає необхідному числу нейронів прихованого шару. Центри кластерів можна використовувати для початкового завдання центрів активаційних функцій нейронів.

Для вирішення другого та третього завдань можна використовувати один із методів глобальної оптимізації, наприклад, генетичний алгоритм (ГА) [51].

Використання ГА передбачає кодування параметрів рядком дійсних чисел – *хромосомою*. Безліч хромосом утворюють популяцію. Кожна хромосома забезпечується оцінкою придатності, тобто відповідності критерію, який може описуватися, наприклад, формулою (5.3).

Алгоритм оптимізації параметрів мережі за допомогою ГА наведено на рис. 5.6. Робота алгоритму закінчується, коли значення помилки досягає прийнятної величини.

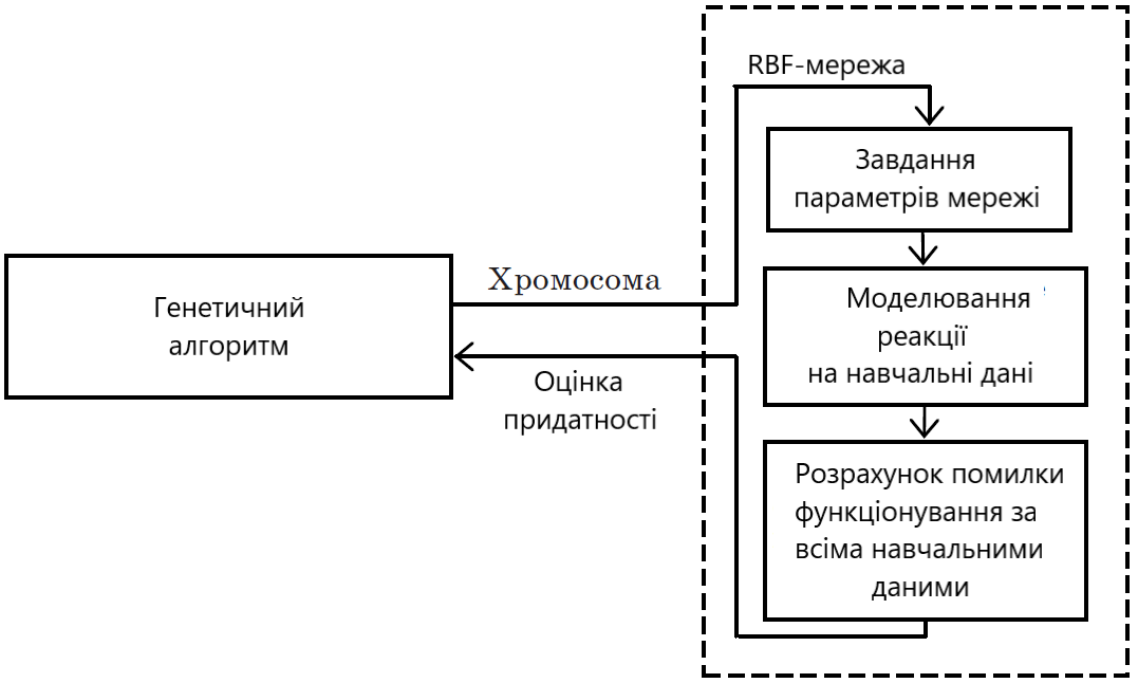


Рисунок 5.6 - Генетична оптимізація параметрів RBF-мережі

5.4. Радіальні нейронні мережі в MatLab

Структура радіального базисного нейрона з входами ***R***, що використовується в MatLab, наведена на рис. 5.7.

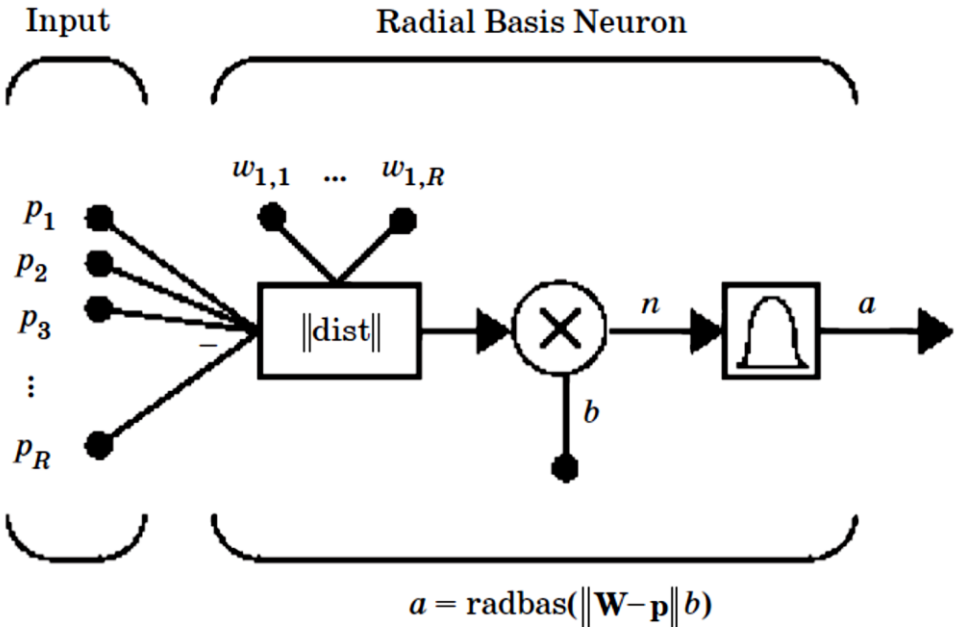


Рисунок 5.7- Радіальний базисний нейрон

Функція dist обчислює евклідову відстань між вектором входу ***P*** та вектором ваг ***W*** нейрона. Зміщення ***b*** служить корекції чутливості нейрона.

Опис його блоку активації наведено на рис. 5.8.

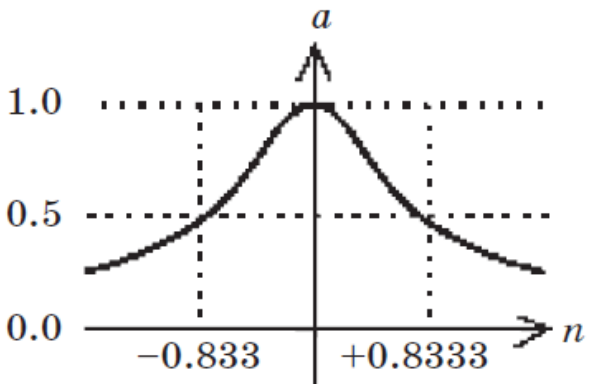


Рисунок 5.8 - Активаційна функція нейрона

Вихід нейрона описується виразом

$$a = radbas(n) = e^{-n^2}.$$

Таким чином, вихід нейрона дорівнює точно одиниці, якщо вектори ***P*** та **W** ідентичні.

Структура RBF-мережі представлена на рис. 5.9. Вона містить S^1 нейронів робочого шару та S^2 нейронів вихідного, що мають лінійні активаційні функції.

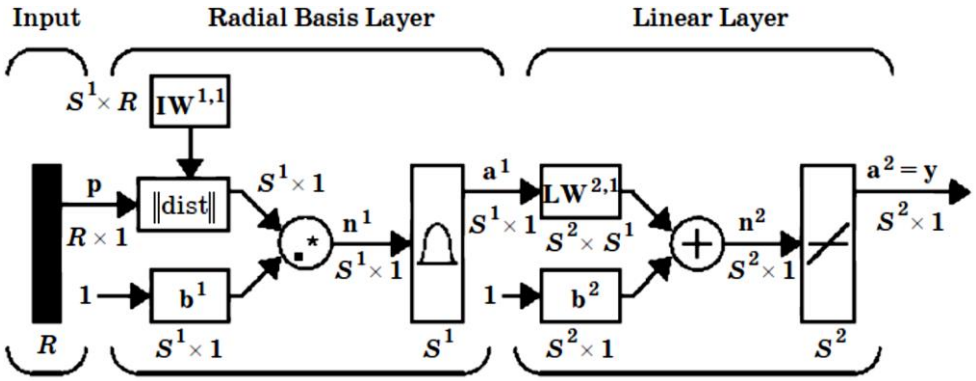


Рисунок 5.9 – Структура RBF-мережі в MatLab

Операції створення RBF-мережі в MatLab оформлені у вигляді функцій newrbe та newrb. Перша дозволяє побудувати радіальну базову мережу з нульовою помилкою, друга – управляти числом нейронів у прихованому шарі.

Функція newrb створює мережу RBF, використовуючи ітеративну процедуру, яка додає по одному нейрону на кожному кроці. Нейрони додаються до прихованого шару до тих пір, поки сума квадратів помилок не стане меншою від заданого значення або не буде використано максимальну кількість нейронів.

Приклад 5. 2. Розглянемо приклад використання функції newrb:

```
P = [0 1 2 3 4 5 6 7 8 9 10]; % визначення навчальних даних
T = [0 1 2 3 4 3 2 1 2 3 4]; % на вході та виході
GOAL = 0.01; % допустима межа помилки
SPREAD = 1; % ширина вікна RBF-функцій
net = newrb(P,T,GOAL,SPREAD); % створення RBF-мережі
%
%Функція newrb створює мережу RBF, використовуючи ітеративну процедуру,
% яка додає по одному нейрону на кожному кроці
%
figure(1), clf, % створення та очищення вікна графіка
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on; % зберігання зображення
X = 0:0.01:10; % завдання безперервного входу RBF-мережі.
Y = sim(net,X); % моделювання роботи мережі
plot(X,Y,'LineWidth',2), grid on % графік роботи RBF-мережі
```

На рис. 5.10 наведено графіки моделювання RBF-мережі.

Дізнатися, скільки утворилося радіальних нейронів, можна за допомогою команди:

```
>> net.layers{1}.size
```

```
ans =     7
```

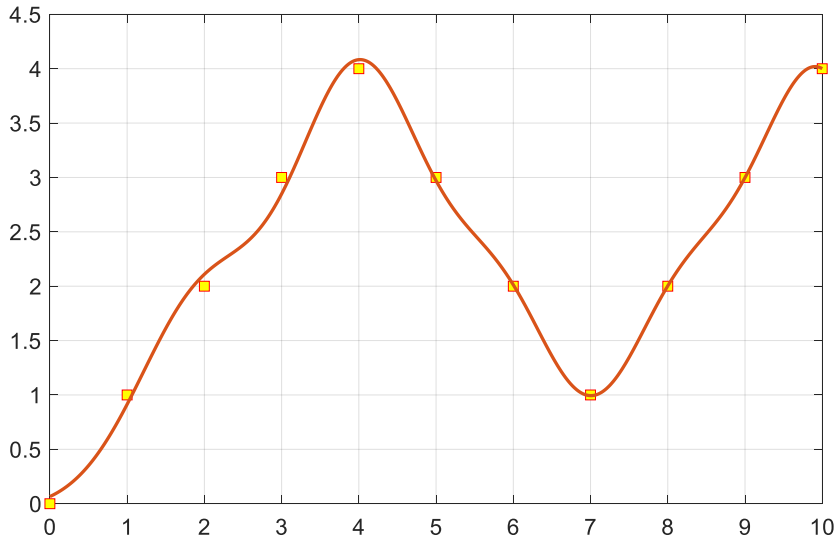


Рисунок 5.10 – Приклад моделювання RBF-мережі в MatLab

При використанні функції newrb число нейронів, що вийшло, залежить від заданої межі помилки і ширини «вікна» активаційної функції. На рис. 5.11 наведено варіант, коли параметр GOAL=0.1. У цьому випадку мережа містить 6 нейронів, і якість апроксимації нижче. На рис. 5.12 показано варіант моделювання при зменшенні зони взаємодії RBF-нейронів (SPREAD=0.5).

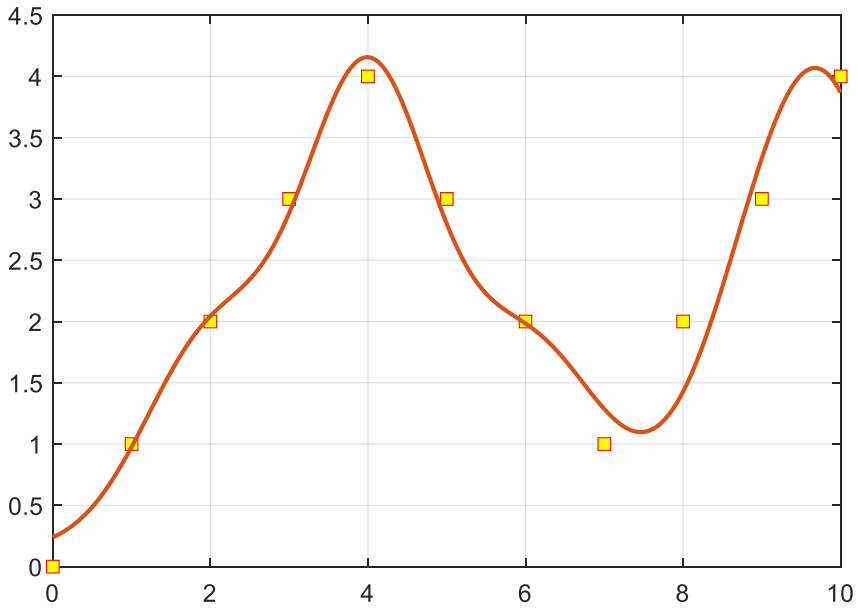


Рисунок 5.11 – RBF-мережа при зниженні вимог до точності

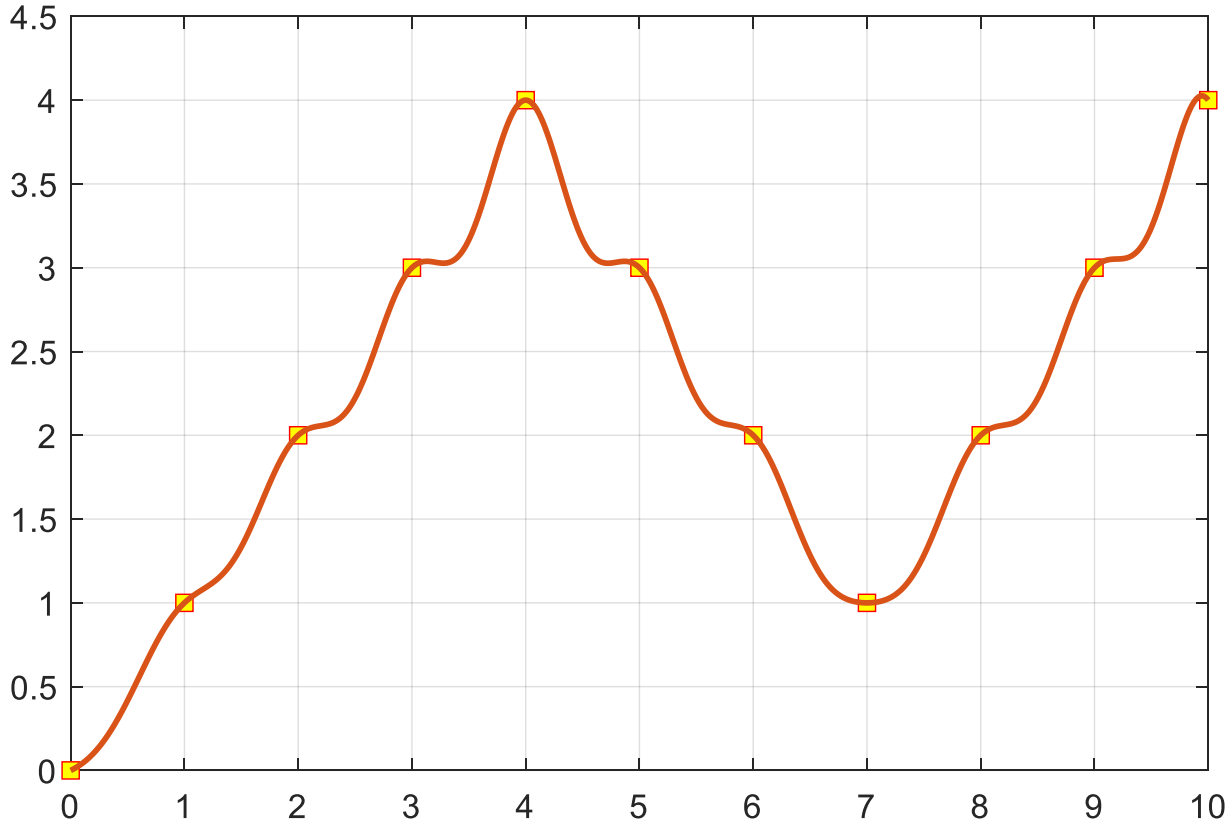


Рисунок 5.12 – RBF-мережа при зменшенні «вікна» активаційної функції

Приклад 5.3. Розглянемо ще один приклад використання функції newrb. Нехай потрібно апроксимувати за точками функцію humps з бібліотеки MatLab:

```
x = 0:.05:2;
y=humps(x);
P=x;
T=y;
GOAL=0.02;
SPREAD= 0.1;
net1 = newrb(P,T,GOAL,SPREAD);
A= sim(net1,P);
plot(x,y,P,A);
xlabel('x, P');
ylabel('y=humps(x), A');
grid
```

Графіки вихідної функції та її апроксимації практично збігаються (див. рис. 5.13).

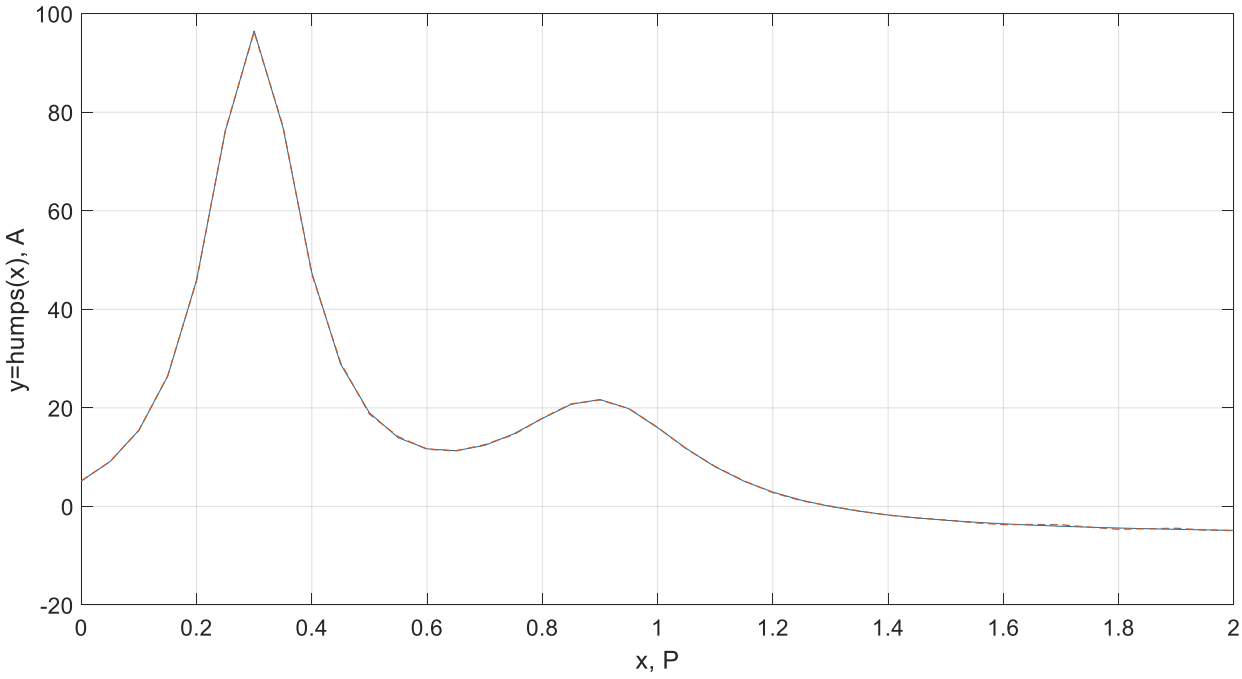


Рисунок 5.13 – Графіки вихідної функції та її апроксимації

У команді створення мережі RBF з нульовою помилкою параметр GOAL не вказується. Наприклад (див. рис. 5.14),

```
P = [0 1 2 3 4 5 6 7 8 9 10];
T = [2 2 2 2 4 4 4 2 2 2 2];
SPREAD = 0.5;
net = newrbe(P,T,SPREAD);
figure(1), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
% Апроксимація
X = 0:0.01:10;
Y = sim(net,X);
plot(X,Y,'LineWidth',2), grid on;
xlabel('P, X');
ylabel('T, Y');
legend('Ф-ція за точками','Апроксимація RBF-мережою')
```

У RBF-мережі з нульовою помилкою число радіальних нейронів вибирається рівним числу елементів навчальної множини, в цьому полягає труднощі використання подібної мережі при великій навчальній вибірці.

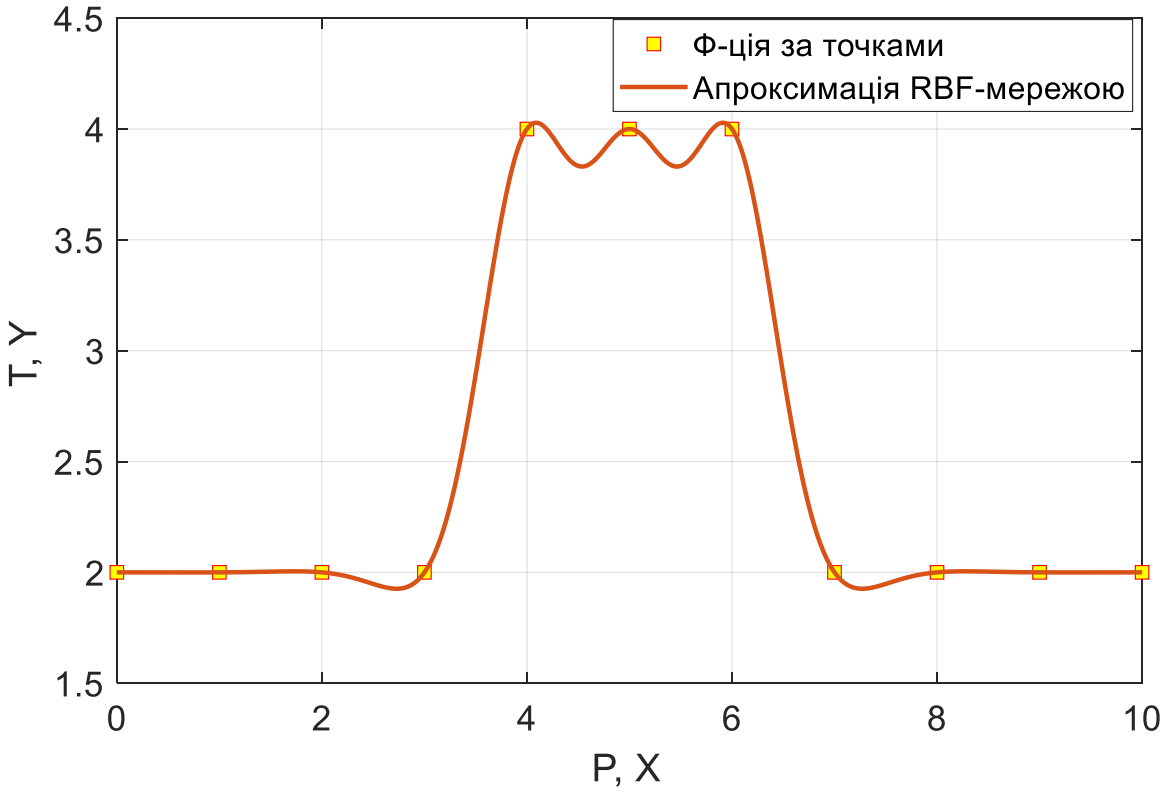


Рисунок 5.14 – Приклад використання RBF-мережі з нульовою помилкою

Приклад 5.4. Розглянемо моделювання системи Ван дер Поля. Нехай динамічна система описується рівнянням Ван дер Поля

$$\frac{d^2y}{dt^2} + (y^2 - 1)\frac{dy}{dt} + y = 0.$$

Цьому рівнянню відповідає схема, зібрана в MatLab Simulink (див. рис. 5.15). На рис. 5.16 показано результат математичного моделювання.

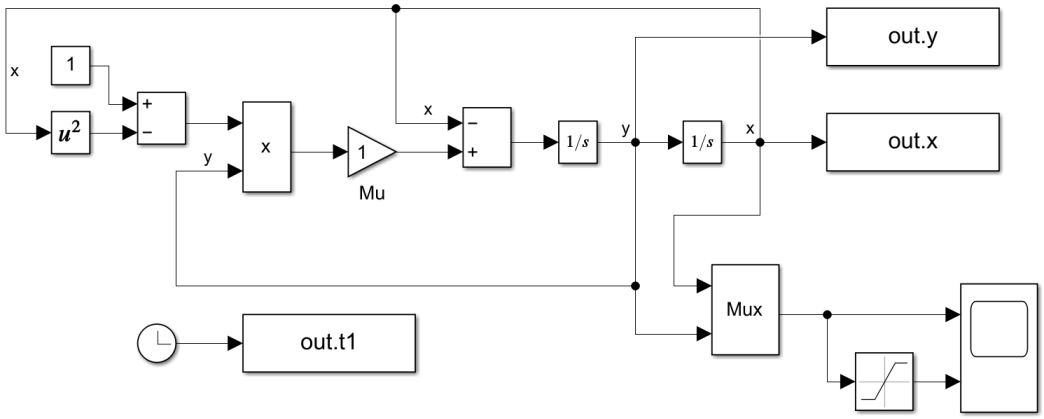


Рисунок 5.15 - Блок-схема рівняння Ван дер Поля

Змінні *x* та *y* описують тут стан динамічної системи (покладемо початковий стан *x* = 2 та *y* = 2). Моделювання з часом 10 с можна запустити командою simOut=sim('VdP1',10), де 'VdP1' – назва mdl-файлу. У цьому випадку буде потрібна така послідовність команд

```
simOut=sim('VdP1',10); % моделювання файлу VdP1.slx
% побудова модельних графіків ф-ції
figure(1);
plot(simOut.t1,simOut.x,simOut.t1,simOut.y);
xlabel('Time, c');
ylabel('x, y');grid;
```

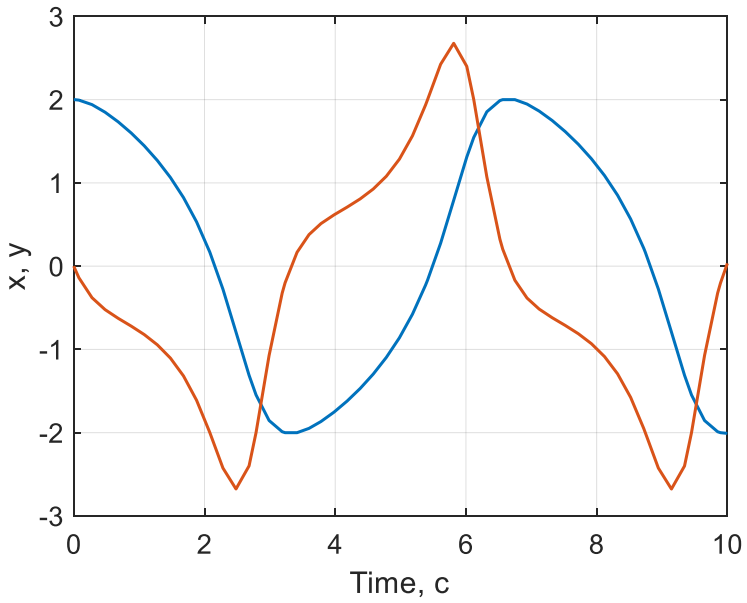


Рисунок 5.16 – Графіки перехідних процесів у системі Ван дер Поля

Для моделювання з використання RBF-мережі буде потрібна така послідовність команд:

```
P = simOut.t1';
T = simOut.y';
net2=newrb(P,T,0.01);
A= sim(net2,P);
T = simOut.x';
net1=newrb(P,T,0.01);
A1= sim(net1,P);
figure(2);
plot(P,A1,P,A)
xlabel('P=Time, c');
ylabel('A, A1');
grid
```

Отриманий результат моделювання, наведений на рисунку 5.18. з використанням RBF-мережі збігається з наведеним графіками на рис. 5.17

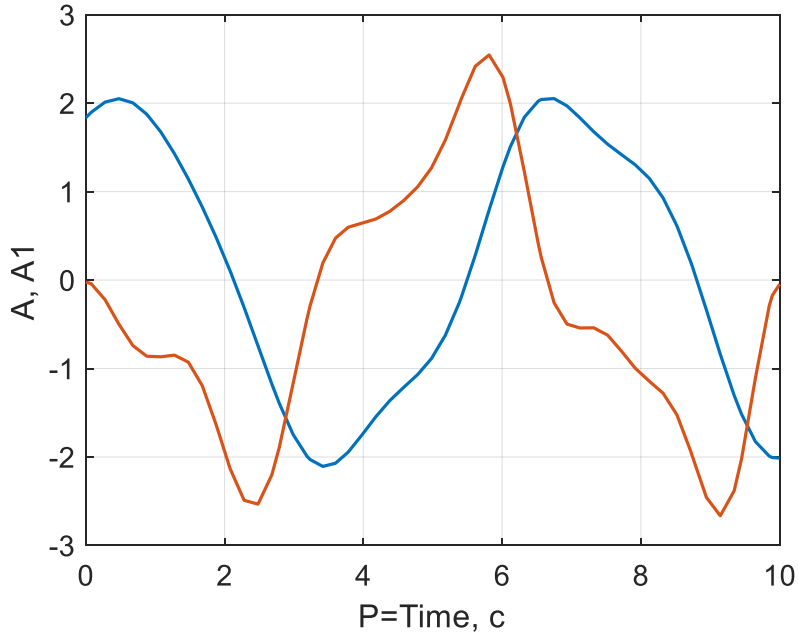


Рисунок 5.16 – Графіки модельованих RBF-мережею перехідних процесів у системі Ван дер Поля

У MatLab описані також різновиди RBF-мереж – мережі GRNN та PNN.

Нейронна мережа GRNN (Generalized Regression Neural Network) призначена для вирішення завдань узагальненої регресії, аналізу часових рядів та апроксимації функцій. Вона має радіально-базисний та спеціальний лінійний шар (див. рис. 5.17). Характерною рисою цих мереж є висока швидкість навчання.

Радіально-базовий шар функціонує так само, як у RBF-мережі з нульовою помилкою. Число нейронів цього шару дорівнює числу елементів ***Q*** навчальної множини. Як початкове наближення для матриці ваг вибирається масив входів ***P***, так що якщо вектор ваги нейрона дорівнює транспонованого вектора входу, вихід функції активації дорівнює одиниці.

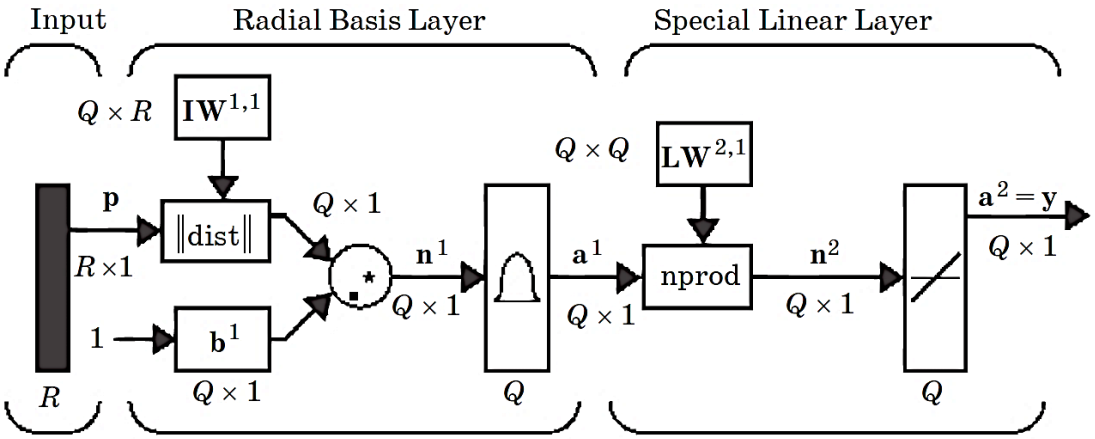


Рисунок 5.17 – Структура GRNN - мережі

Другий шар - це лінійний шар з числом нейронів, також рівним ***R***, причому як початкове наближення для матриці ваг ***LW*{2,1}** вибирається масив виходів ***T***.

Блок normprod служить для обчислення нормованого скалярного добутку рядка масиву ваги вихідного шару та вектору виходу радіально-базисного шару.

Таким чином, якщо вектор входу близький до одного з векторів входу P_i з навчальної множини, то значення *i*-го виходу радіально-базового шару буде близько до одиниці, і вихід другого шару буде близький до T_i.

Якщо параметр впливу SPREAD малий, то діапазон вхідних значень, який реагує нейрон прихованого шару, виявляється малим. Зі збільшенням параметра SPREAD кілька нейронів реагує на значення вектора входу. Тоді на виході мережі формується вектор, що відповідає середньому кількох цільових векторів, відповідних вхідним векторам навчальної множини, близьких до вектора входу.

Приклад 5.5. з метою порівняння результатів апроксимації з застосуванням RBF-мережі та нейронної мережі GRNN застосуємо навчальну вибірку та результати навчання з прикладу 5.3. У MatLab нейронна мережа GRNN задається командою newgrnn.

```
P = [0 1 2 3 4 5 6 7 8 9 10];
T = [2 2 2 2 4 4 4 2 2 2 2];
SPREAD = 0.5;

net = newgrnn(P,T);
figure(1), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
X = 0:0.01:10; Y = sim(net,X); plot(X,Y,'LineWidth',2), grid on

net.layers{1}.size
```

Результат апроксимації, наведений на рисунку 5.18, суттєво відрізняється від наведеного на рис. 5.14 (при однаковій навчальній множині).

Нейронні мережі PNN (Probabilistic Neural Network) призначені для вирішення ймовірнісних завдань, у тому числі і класифікації.

У такій мережі перший шар містить радіально-базисні функції. Другий шар називається *шаром конкуренції* (див. рис. 5.19).

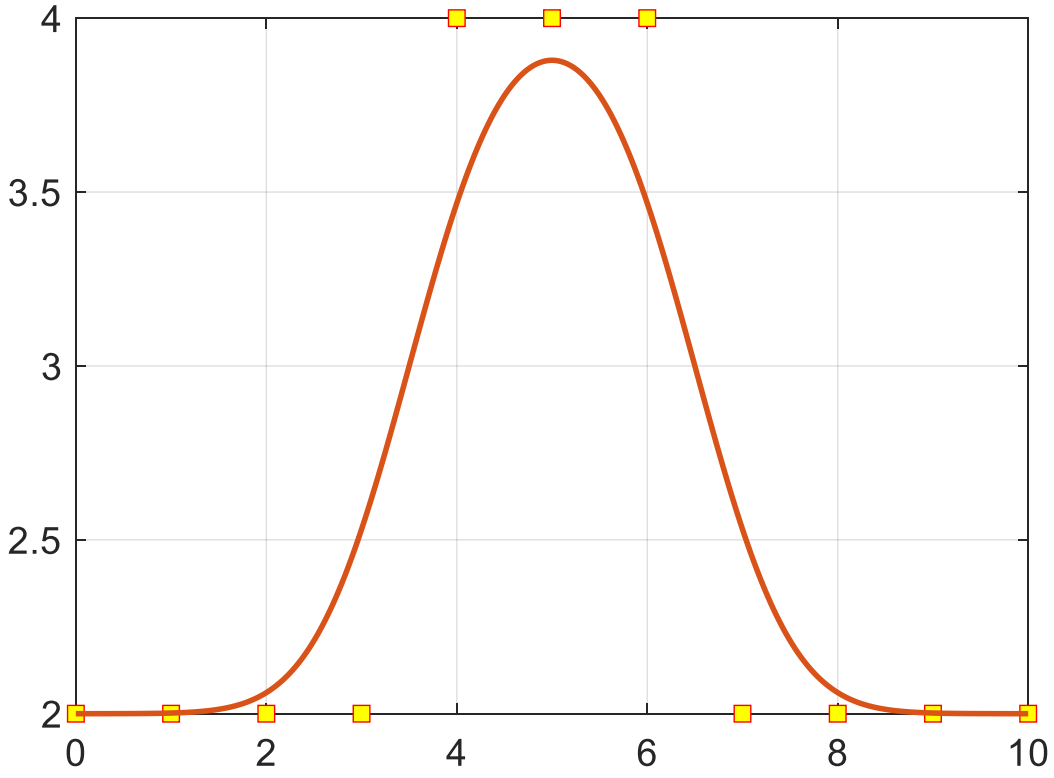


Рисунок 5.18 – Графік апроксимації за допомогою GRNN-мережі

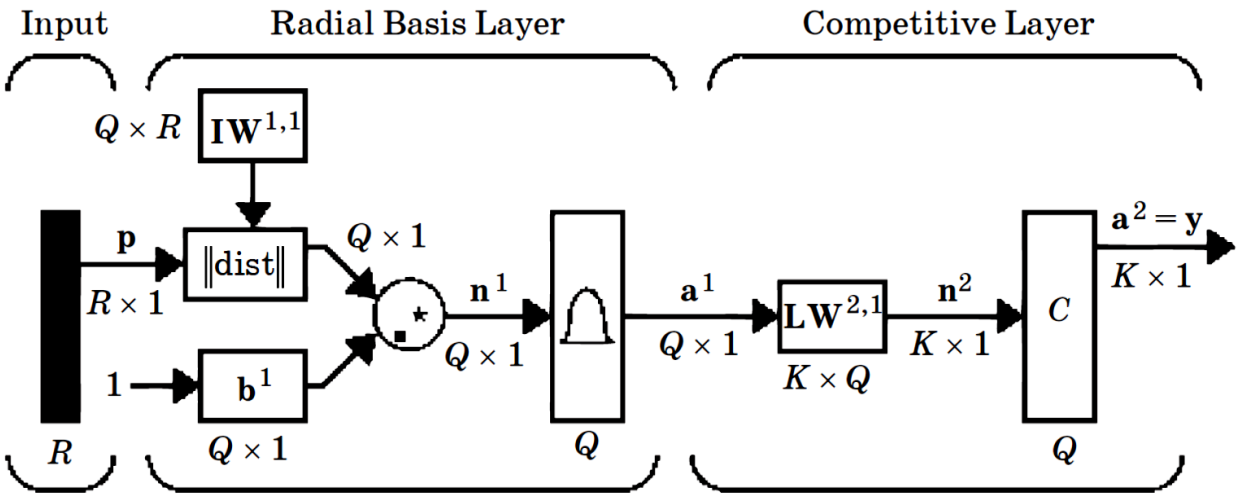


Рисунок 5.19 – Структура нейронної мережі PNN

Радіально-базовий шар функціонує аналогічно до відповідного шару GRNN-мережі.

Шар конкуренції обчислює ймовірність приналежності вектора до того чи іншого класу, та був вибирає клас із найбільшою ймовірністю.

Навчальна множина містить ***Q*** пар векторів вхід-ціль. Мається ***K*** класів, яких може належати вхідний вектор.

Приклад 5.6. Нехай, наприклад, розглядаються сім векторів та три класи приналежності:

```
P = [0 0;1 1;0 3;1 4;3 1;4 1;4 3]';
Tc = [1 1 2 2 3 3 3];
```

З цих даних може бути утворена матриця зв'язності ***T*** розміром ***K*×*Q***, що складається з нулів і одиниць, рядки якої відповідають класам власності, а стовпці – векторам входу. Таким чином, якщо елемент *T*(*i*, *j*) матриці зв'язності дорівнює одиниці, це означає, що *j* -й вхідний вектор належить до класу *i*.

У розрідженій формі ця матриця цільової функції має вигляд

```
T = ind2vec(Tc); % Матриця цільової ф=ції у розрідженій формі

T =

   (1,1)        1
   (1,2)        1
   (2,3)        1
   (2,4)        1
   (3,5)        1
   (3,6)        1
   (3,7)        1
```

У повній формі ця матриця цільової функції має вигляд

```
T1=full(T); % Матриця у повній формі

T1 =

     1     1     0     0     0     0     0
     0     0     1     1     0     0     0
     0     0     0     0     1     1     1
```

Масиви ***P*** і ***T*** задають навчальну множину, що дозволяє виконати формування мережі. Масив T – задає матрицю навчальної функції у розрідженої формі, а T1 – у повній.

```
net = newpnn(P,T);
Y = sim(net,P);
net1 = newpnn(P,T1);
Y1 = sim(net1,P);
```

Для перевірки роботи мережі можна подавати на її вхід довільні вектори

```
%Перевірка роботи мережі (подаємо на її вхід довільні вектори)
```

```
P = [0.1 0.5;1.2 1.3;4 4]';
%
% При матриці цільової ф=ції у розрідженій формі
%
Yp = sim(net,P)
Ycp = vec2ind(Yp)
%
% При матриця цільової ф=ції у повній формі
%
Yp1 = sim(net1,P)
Ycp1 = vec2ind(Yp1)
```

Результат при повний формі та розрідженій формі цільової функції співпадає

```
Yp1 =
     1     1     0
     0     0     0
     0     0     1
Ycp1 =
     1     1     3
```

Таким чином, два перших вектору приналежать до класу 1, а третій вектор – до класу 3.

Розглянуті варіанти радіально-базових мереж використовують фундаментальний принцип нелінійного перетворення вхідного простору на прихований простір вищої розмірності. У просторі вищої розмірності з більшою ймовірністю можлива лінійна роздільність вхідного простору, а також підвищується можливість гладкої апроксимації вхід-вихідних даних.

Проста структура RBF-мереж, що містить лише один шар прихованих нейронів, уможливлює прямий розрахунок ваги мережі. У цьому полягає перевага даних мереж порівняно з іншими типами НМ, які використовують трудомісткі алгоритми навчання. Втім, RBF-мережа - перенавчається.

5.5 Радіальні нейронні мережі та нечіткі системи

Можна показати, що робота мережі RBF функціонально відповідає роботі системи нечіткого логічного висновку. Описи математичних основ нечітких логічних систем і нечітких регуляторів можна знайти, наприклад, у [38]. Далі розглянемо типову структуру системи нечіткого логічного висновку з n вхідними змінними (посилками), m правилами та одним висновком. Цю структуру можна як ШНМ прямого поширення, як показано на рис. 5.20 [52].

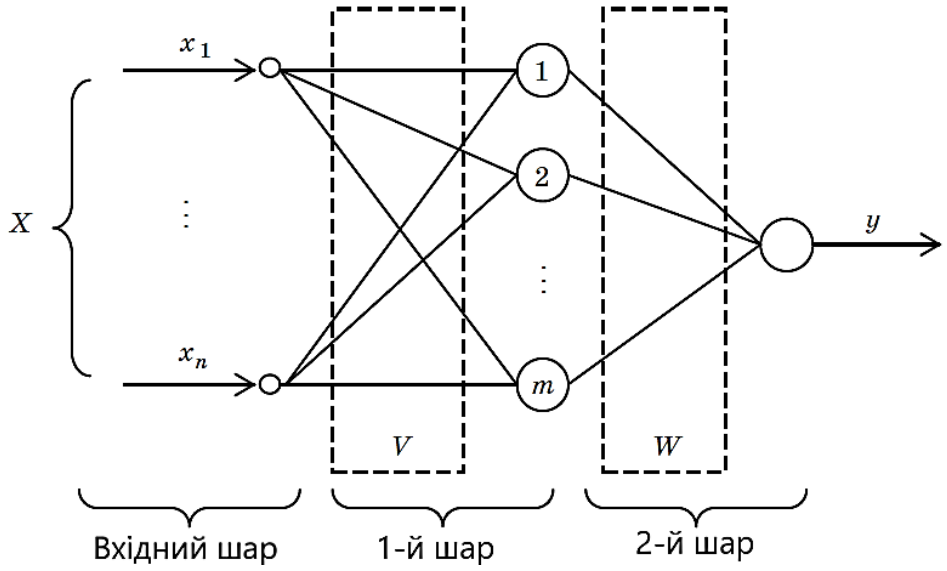


Рисунок 5.20 - Система нечітких правил як двошарова ШНМ

Вузли вхідного шару просто передають вхідні сигнали вузлам 1-го (прихованого) шару, які обчислюють максимальне відхилення вхідного сигналу відповідної посилки правила.

Ваги *V* описують посилки правил. Кожен *i*-й нейрон має свій набір ваг $\{v_{ji}\}$, j = {1, 2, ..., *n*}, які відповідають значенням посилок правила. Вектори *X* та *V* зазвичай нормалізуються.

Кожен нейрон 1-го шару обчислює подібність вхідного вектора та свого вектора посилок, у результаті визначається ступінь запуску відповідного правила.

Для вимірювання подібності *X* і *V* може бути використане обчислення:

- скалярного твору векторів:

$$R = XV = \sum_{i=1}^{n} x_i v_i ;$$

- максимальне відхилення

$$R = \max(\lceil x_1 - v_1 \rceil, \lceil x_2 - v_2 \rceil, \cdots, \lceil x_n - v_n \rceil) ;$$

- евклідова відстань між векторами

$$R = \sqrt{(x_1 - v_1)^2 + (x_2 - v_2)^2 + \cdots + (x_n - v_n)^2}.$$

Останній варіант відповідає операції обчислення подібності, що виконується RBF-нейроном.

Потім отримана величина використовується як аргумент функції приналежності, яка може мати вигляд, наведений на рис. 5.21.

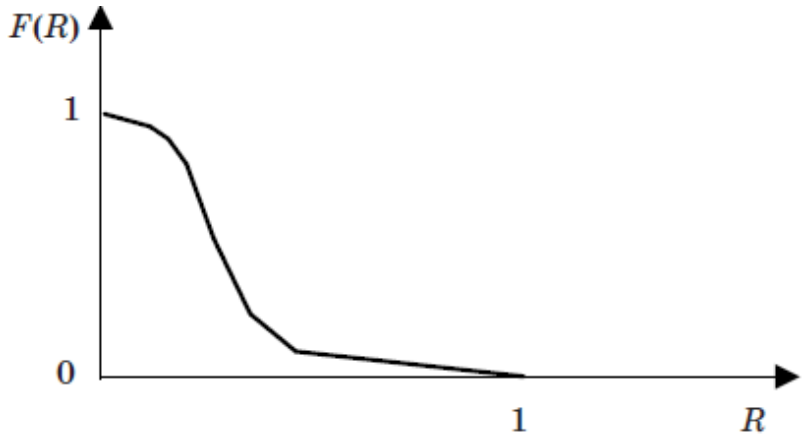


Рисунок 5.21 – Варіант опису функції приналежності

Отже, робота 1-го шару структури на рис. 5.20, що описує нечітку систему виведення, практично відповідає роботі радіально-базового шару. Усі відмінності стосуються деталей реалізації.

Нейрон 2-го шару структури наведеної на рис. 5.20 виконує операцію дефазифікації. Кожна вага вектора *W* відповідає висновку правила, і вихідний сигнал є зваженою сумою виходів нейронів 1-го шару:

$$y = w_1 \frac{F(R_1)}{\sum_{i=1}^{m} F(R_i)} + w_2 \frac{F(R_2)}{\sum_{i=1}^{m} F(R_i)} + \cdots + w_n \frac{F(R_m)}{\sum_{i=1}^{m} F(R_i)}.$$

Якщо нормалізувати вихід RBF-мережі, то вийде такий самий результат.

Таким чином, RBF-мережа функціонально еквівалентна системі нечітких правил

6 МОДЕЛІ АСОЦІАТИВНОЇ ПАМ'ЯТІ

6.1. Нейрона мережа Елмана

Нейрона мережа Елмана належить до класу частково рекурентних ШНМ (див. рис. 6.1).

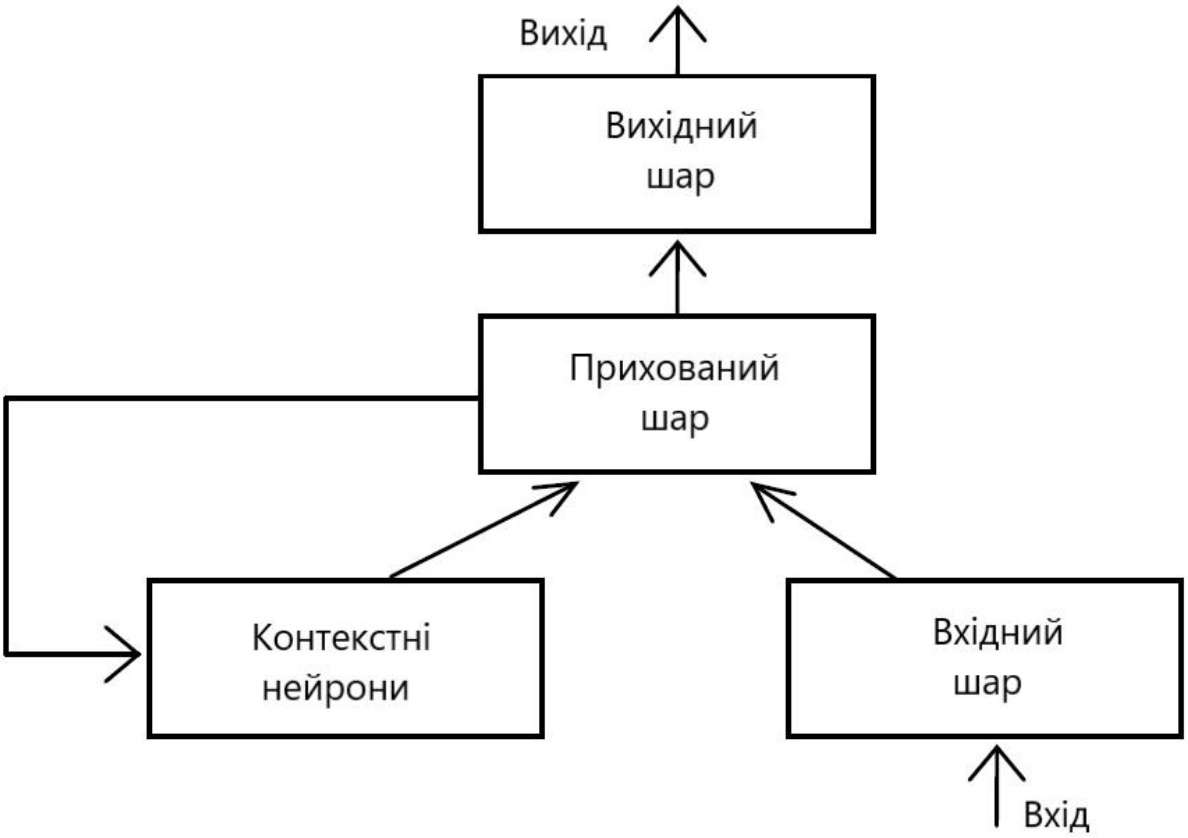


Рисунок 6.1 – Структура ШНМ Елмана

Багато практичних завдань можна перетворити на завдання розпізнавання часових послідовностей. Як простий приклад розглянемо функцію XOR. Як було показано, для її реалізації потрібно використовувати двошарову мережу прямого поширення. Альтернативою є опис функції XOR у часовій області як послідовності бітів. Наприклад,

1 0 1 0 0 0 0 1 1 1 1 0 1 0 1…

Тут кожен третій біт є логічною функцією перших двох бітів.

Таким чином, поведінка ШНМ, що реалізує функцію XOR, повинне полягати в передбаченні наступного біта послідовності:

Вхід: 1 0 1 0 0 0 0 1 1 1 1 0 1 0 1

Вихід: 0 1 0 0 0 0 1 1 1 1 0 1 0 1 ?

Для обробки тимчасових послідовностей у роботі [53] було запропоновано структуру ШНМ, наведену на рисунку 6.2.

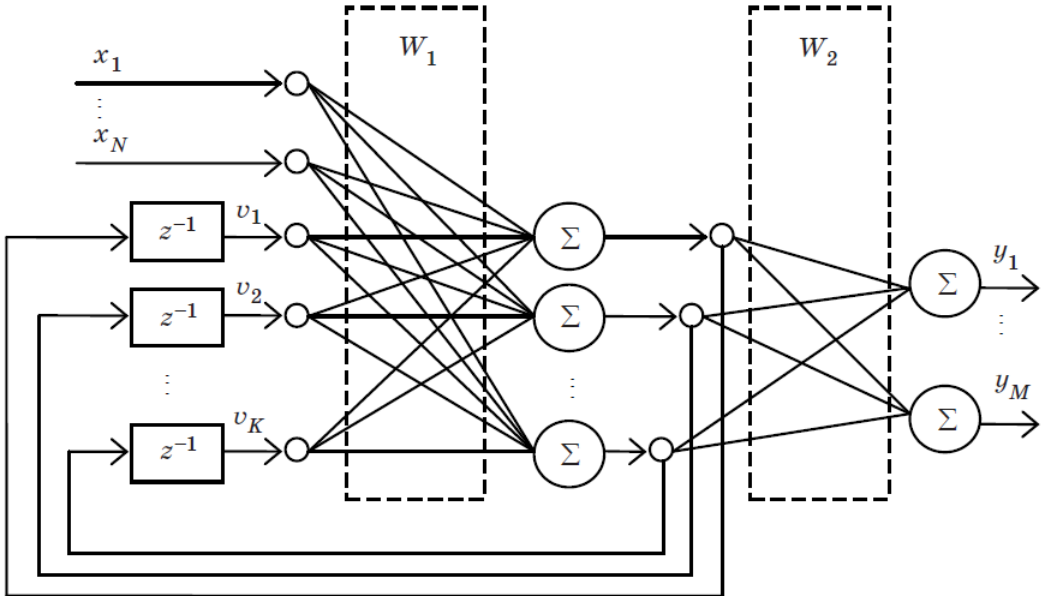


Рисунок 6.2 – Нейронна мережа Елмана

Нейронна мережа Елмана складається з *N* входів, *K* нейронів прихованого шару, охоплених зворотними зв'язками через елементи затримки z^{-1}, *M* нейронів вихідного шару.

Вектор стану вхідного шару мережі на момент часу kT_0 (T_0 – крок дискретизації за часом) має вигляд

$$X(k) = [x_0(k), x_1(k), \cdots, x_n(k), v_1(k-1), \cdots, v_K(k-1)],$$

де $x_0(k), x_1(k), \cdots, x_n(k)$ – вектор вхідного сигналу; $v_1(k-1), \cdots, v_K(k-1)$ – стан нейронів прихованого шару.

Стан нейронів прихованого шару визначається виразом

$$v_i(k) = f_i^1\big(u_i(k)\big), \qquad u_i(k) = \sum_{j=0}^{N+K} w_{ij}^1 v_j(k),$$

де w_{ij}^1 – синаптичні вагові коефіцієнти; $f_i^1\big(u_i(k)\big) = h\big(u_i(k)\big)$ – функція активації нейронів прихованого шару.

Аналогічно для вихідного шару:

$$y_i(k) = f_i^2\big(g_i(k)\big), \qquad g_i(k) = \sum_{j=0}^{M} w_{ij}^2 v_j(k),$$

де $y_0(k)$, $y_1(k)$, …, $y_M(k)$ – вектор вихідного сигналу мережі.

У вхідному шарі двошарової мережі Елмана використовується активаційна функція гіперболічного тангенсу tansig, у вихідному шарі – лінійна функція purelin. Таке поєднання передаточних функцій дозволяє

максимально точно апроксимувати функції кінцевого числа точок розриву. Для цього необхідно також, щоб вихідний шар мав досить велике число нейронів. Всі ці верстви Елмана мають усунення.

При навчанні мережі Елмана ваги вихідного шару піддаються такій самій корекції, як і ваги персептрона. Формули уточнення ваги прихованого шару мережі Елмана більш складні через наявність зворотних зв'язків між прихованим і контекстним шарами [54].

У MatLab мережі Елмана створюються функцією

net = newelm(PR, [S1, S2, …, SN], {TF1, TF2, …, TFN}, BTF, BLF, PF),

де PR – масив розміром R×2 мінімальних та максимальних значень для R векторів входу; S1, S2, …, SN – число нейронів у шарах; TF1, TF2, …, TFN – функції активації у шарах (за замовчуванням tansig); BTF – навчальна функція, що реалізує метод зворотного розповсюдження (за замовчуванням traingdx); BLF – функція налаштування, що реалізує метод зворотного розповсюдження (за замовчуванням learngdm); PF – критерій якості навчання (за замовчуванням mse).

Мережі Елмана орієнтовані моделювання часових рядів.

Приклад 6.1. Нехай нейронна мережа має у відповідь видавати послідовність бітів, у якій з'являється одиничний біт, якщо у вихідній послідовності зустрілося дві одиниці поспіль.

Задамо випадкову послідовність із 20 бітів командою

```
P = round (rand (1, 20));
%Тоді цільовий вектор можна описати командою
T = [0 (P(1:end-1)+P(2:end) == 2)];
% Для навчання створимо масиви осередків:
Pseq = con2seq(P);
Tseq = con2seq(T);
% Мережа Елмана створюється командою
net = newelm ([0 1], [10, 1], {'tansig', 'logsig'});
% Завдання параметрів та запуск навчання:
net.trainParam.goal = 0.001;
net.trainParam.epochs = 1000;
net = train (net, Pseq, Tseq);
% Перевірка:
Y = sim (net, Pseq)
```

Масив випадкової послідовності

```
P =
  Columns 1 through 13
    1     0     0     0     0     0     1     1     1     0     0     0     1
  Columns 14 through 20
    0     0     0     1     1     1     0
```

Масив цільового вектору

```
T =
  Columns 1 through 13
    0     0     0     0     0     0     0     1     1     0     0     0     0
  Columns 14 through 20
    0     0     0     0     1     1     0
```

Результати перевірки

```
Y =
  1×20 cell array
  Columns 1 through 5
    {[0.0575]}    {[0.0045]}    {[0.0248]}    {[0.9618]}    {[0.9691]}
  Columns 6 through 10
    {[0.9469]}    {[0.9443]}    {[0.0413]}    {[0.0115]}    {[0.0035]}
  Columns 11 through 15
    {[0.0086]}    {[0.0280]}    {[0.0029]}    {[0.0011]}    {[0.0046]}
  Columns 16 through 20
    {[0.0445]}    {[0.9828]}    {[0.0377]}    {[0.0089]}    {[0.0027]}
```

Ступінь відповідності досить високий. Результати моделювання НМ Елмана наведені на рисунку 6.3.

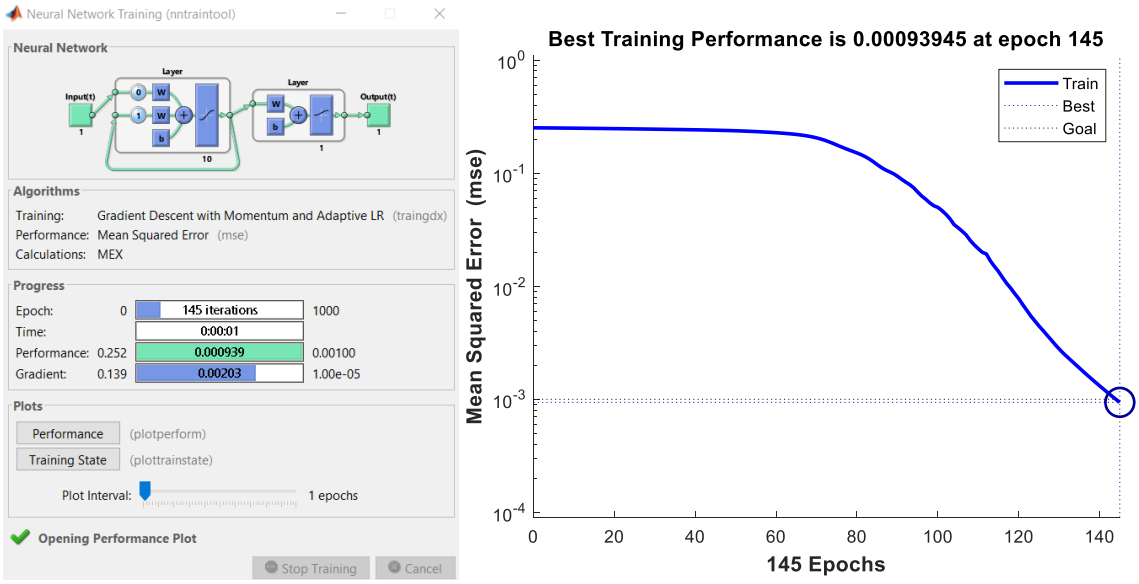


Рисунок 6.3 – Результати моделювання НМ Елмана

Проведемо перевірку для випадкової послідовності

```
>> P = round (rand (1, 20))
P =
  1 0 1 0 1 0 0 1 1 0 1 1 0 0 0 0 1 1 1 1
>> Pseq = con2seq(P);
>> Y = sim(net, Pseq)
```

Результати перевірки

```
Yseq =
  1×20 cell array
  Columns 1 through 5
    {[0.0883]}    {[0.9638]}    {[0.6921]}    {[0.8100]}    {[0.0104]}
  Columns 6 through 10
    {[0.0057]}    {[0.1381]}    {[0.0057]}    {[0.0029]}    {[0.0053]}
  Columns 11 through 15
    {[0.0049]}    {[0.0040]}    {[0.0040]}    {[0.0041]}    {[0.0040]}
  Columns 16 through 20
    {[0.0011]}    {[0.0909]}    {[0.7111]}    {[0.0708]}    {[0.0876]}
```

Приклад 6.2. Використання мережі Елмана для визначення амплітуди гармонійного сигналу.

Визначаються дві синусоїди, амплітуди яких різняться вдвічі:

```
p1 = sin(1:20);
p2 = sin(1:20)*2;
%
% Цільові вектори заповнюються відомим значенням амплітуди кожної
синусоїди:
t1 = ones(1,20);
t2 = ones(1,20)*2;
%
% Формується комбінація синусоїд та відповідних цільових векторів:
p = [p1 p2 p1 p2];
t = [t1 t2 t1 t2];
%
% Формуються масиви осередків:
Pseq = con2seq(p);
Tseq = con2seq(t);
%
% Створюється та навчається мережа Елмана.
%net = elmannet(1:2,10);
%net = newelm ([-2 2], [10, 1], {'tansig', 'logsig'});
net = newelm(Pseq,Tseq,10);
% Завдання параметрів та запуск навчання:
net.trainParam.goal = 0.01;
net.trainParam.epochs = 1000;
net = train (net, Pseq, Tseq);
view(net);
%
% Перевірка на відтворення навчальних даних, побудова графіку:
a = sim(net,Pseq);
a1 = seq2con(a);
x = 1:80;
```

```
figure(1);
plot(x,a1{1,1})
```

Результат побудови НМ Елмана для визначення амплітуди гармонійного сигналу наведено на рисунку 6.4, а графік перевірки на відтворення навчальних даних на рисунку 6.5

Перевіримо роботу мережі за інших амплітуд сигналів:

```
p1 = sin(1:20)*1.5;
p2 = sin(1:20)*2.5;
p = [p1 p2 p1 p2 p2];
Pseq = con2seq(p);
a = sim(net,Pseq);
a1 = seq2con(a);
x = 1:100;
figure(2);
plot(x,a1{1,1});
grid
```

Результат перевірки роботу мережі за інших амплітуд сигналів наведено на рисунку 6.6.

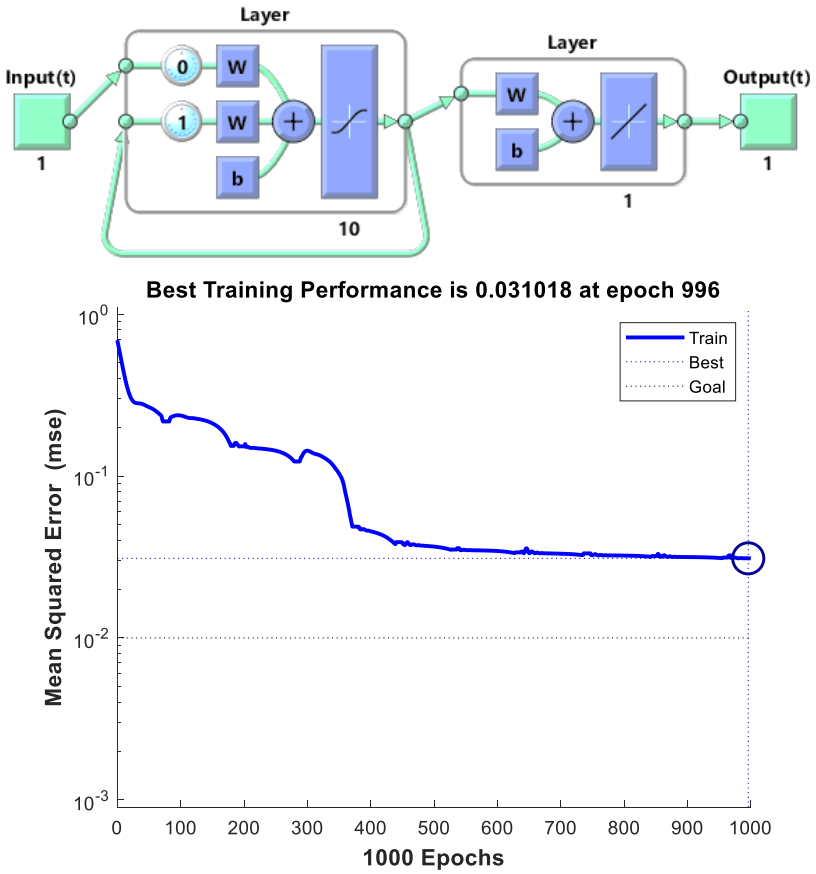


Рисунок 6.4 – Результат побудови НМ Елмана для визначення амплітуди гармонійного сигналу

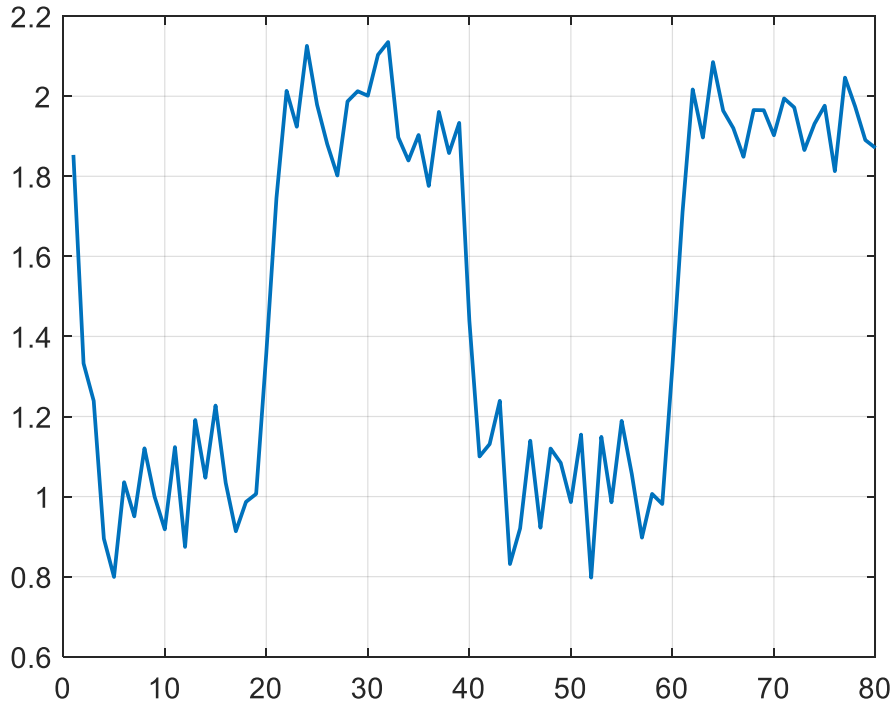


Рисунок 6.5 - Графік перевірки на відтворення навчальних даних

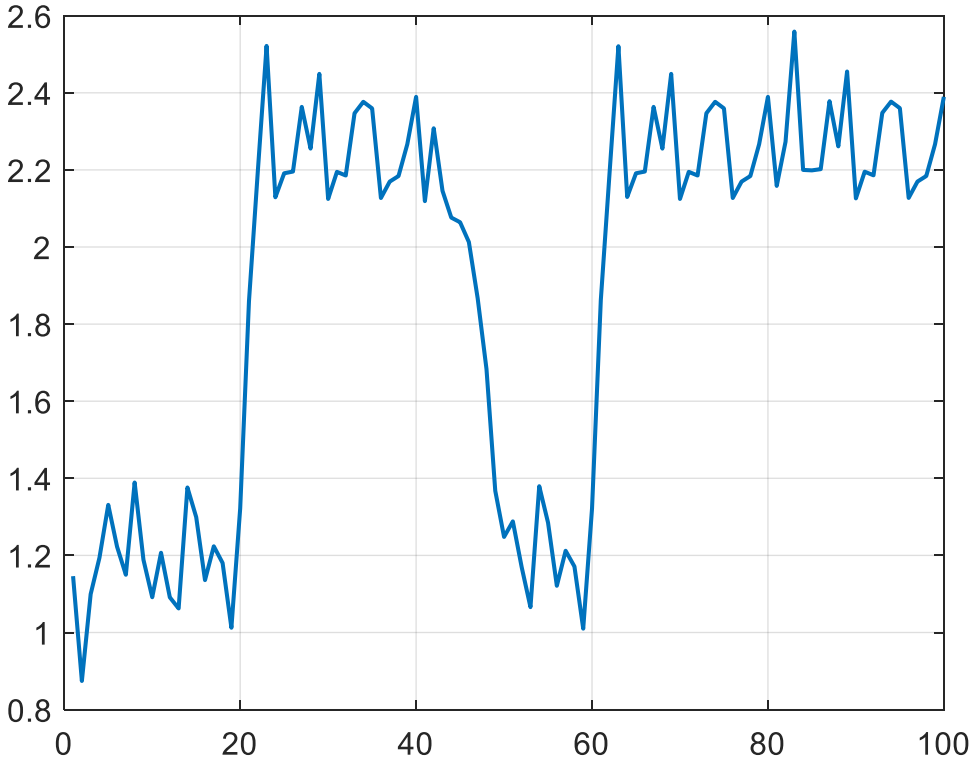


Рисунок 6.6 - Графік перевірки для зміненої амплітуди сигналу

6.2 Мережа Хопфілда

Нейронна мережа Хопфілда - одношарова мережа, в якій кожен нейрон має зв'язки з іншими нейронами (див. рис. 6.7).

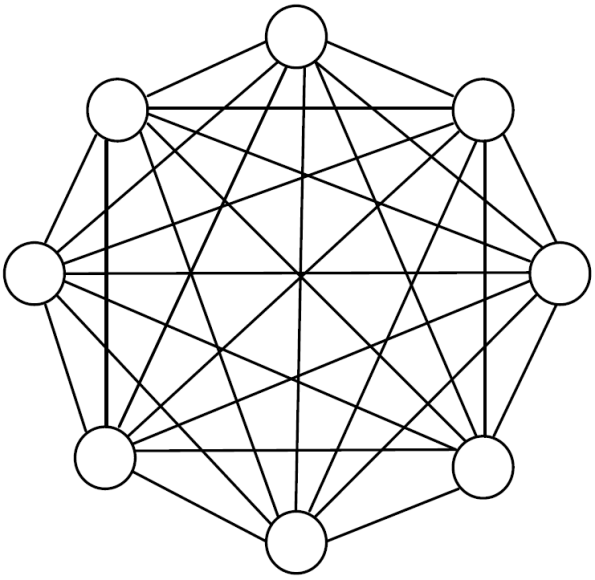


Рисунок 6.7 – Мережа Хопфілда з восьма нейронами

Мережі Хопфілда мають зворотні зв'язки які є динамічними (рекурентними), оскільки після отримання кожного нового вхідного сигналу починається перехідний процес, який закінчується встановленням постійного виходу або продовжується нескінченно довго. Тому проблема стійкості мережі зі зворотними зв'язками може мати велике значення під час вирішення прикладних завдань.

При подачі на вхід стійкої мережі нового вхідного вектора ШНМ переходить від стану до стану, доки не стабілізується. Можна сміливо сказати, що стійкі точки (атрактори) утворюють зони тяжіння (див. рис. 6.8). Крім цільових атракторів у мережі можуть мати місце хибні атрактори, яким не відповідає жодний образ.

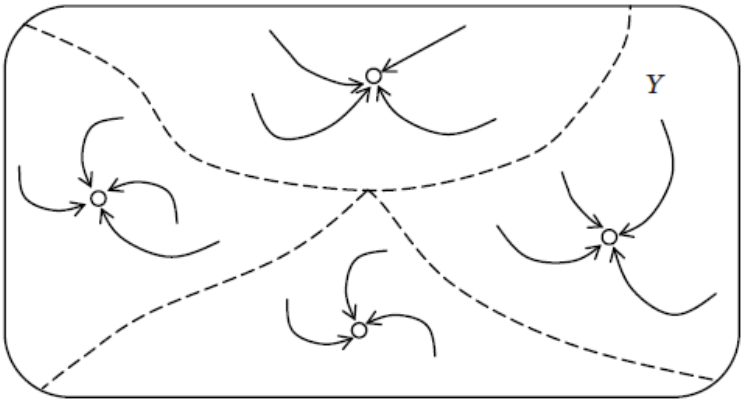


Рисунок 6.8 - Зони тяжіння та атрактори у просторі станів

При подачі на вхід мережі частково неправильного вектора вхідного мережа стабілізується в стані, найближчому до бажаного. Для цього ваги мережі слід вибирати так, щоб утворювати стан стійкої рівноваги для кожного вхідного образу.

Кожен нейрон мережі Хопфілда отримує вхідний сигнал, і навіть сигнали з виходів решти нейронів (рис. 6.9).

Вихід нейрона змінюється за формулою

$$y_i^{t+1} = F\left(\sum_i w_j y_i + x_i\right).$$

де x_i – вхідний сингал.

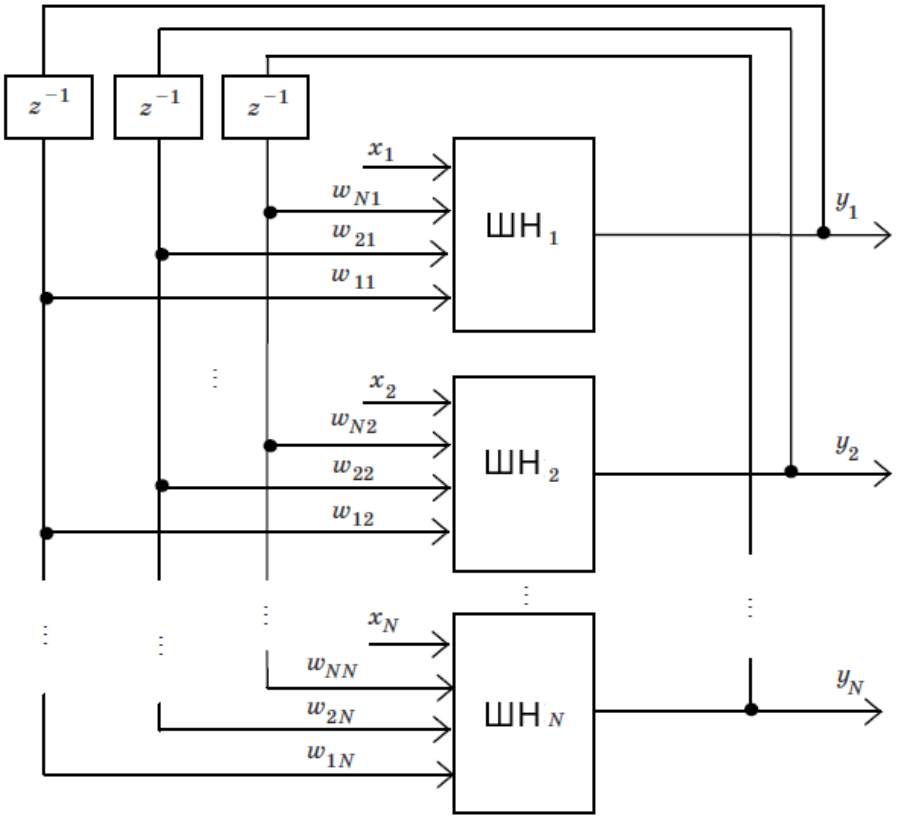


Рисунок. 6.9 – Структура мережі Хопфілда

Як активаційна використовується порогова функція з порогом *T*:

$$y_i^{t+1} = \begin{cases} y^t, & \text{якщо } \sum_i w_j y_i + x_i = T, \\ +1, & \text{якщо } \sum_i w_j y_i + x_i > T, \\ -1, & \text{якщо } \sum_i w_j y_i + x_i < T. \end{cases}$$

Зазвичай обирають нульовий поріг.

Значення (–1) – кодує бінарний 0.

Стан мережі – це двійкове число завдовжки *N* біт:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix},$$

та система з m нейронами буде мати 2^m станів.

Докази стійкості ШНМ із зворотними зв'язками довільного виду на даний час немає, але є теорема про стійкість підмножини таких мереж.

Відповідно до цієї теореми мережа із зворотними зв'язками стійка, якщо її матриця ваг W симетрична: $w_{ij} = w_{ji}$, а елементи головної діагоналі нульові: $w_{ii} = 0$.

У мережі Хопфілда ваги задаються, і навчання як таке відсутнє. Обчислення ваг проводиться за формулою

$$w_{ij} = \begin{cases} \sum_{k=1}^{m} x_j^k x_i^k, & i \neq j, \\ 0, & i = j. \end{cases}$$

де x_j^k – j-я компонента k-го вектора, що запам'ятовується; m – загальна кількість образів, що запам'ятовуються.

Інакше кажучи, ваговий масив W може бути знайдений шляхом обчислення твору кожного вектора, що запам'ятовується, з самим собою і підсумовуванням отриманих матриць розміром $n \times n$:

$$W = \sum_{k=1}^{m} X_k^T X_k - E.$$

Неважко розрахувати, що ваги мережі Хопфілда дорівнює $N^2 - N$. Прикладом, при довжині вхідного вектора 120 бітів отримуємо 1202 – 120 = 14280.

При великій довжині векторів, що запам'ятовуються, доцільно використовувати формулу

$$W = \frac{1}{n} \sum_{k=1}^{m} X_k^T X_k - E.$$

При такому підході реалізується правило навчання Хебба: якщо два нейрони одночасно збуджені, зв'язок між ними посилюється.

Приклад 6.3. Нехай потрібно запам'ятати образ

$$X = [1 \quad 1 \quad 1 \quad -1]^T.$$

Матриця ваг:

$$W = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} [1 \quad 1 \quad 1 \quad -1] - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}.$$

Нехай на вхід мережі надходить зашумлений вектор $Y(0)$. Тоді

$$Y(1) = sgn[WY(0)] = sgn \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = sgn \begin{bmatrix} 1 \\ 3 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix},$$

$$Y(2) = sgn[WY(1)] = sgn \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = sgn \begin{bmatrix} 3 \\ 3 \\ 3 \\ -3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}.$$

Таким чином мережа відновлює запам'ятований образ.

Алгоритм роботи мережі Хопфілда може бути описаний так:

1. Задається вхідний образ (неповний, спотворений): Y при $t = 0$;

2. Виконуються асинхронні обчислення з нейронної мережі (ітераційне правило):

$$Y_i^{t+1} = sgn\left[\sum_{j=1}^{n} w_{ij} Y_i^t\right].$$

3. Обчислення виконуються до того часу, доки настане рівновага, тобто доки всі виходи стануть постійними. Цей вектор набуває таким чином відповідність вхідному вектору Y.

Для гарної роботи мережі Хопфілда необхідно, щоб образи, що запам'ятовуються, були мало схожі один на одного, тобто слабо корельовано (або ортогональні).

Міра корельованих двійкових векторів X_j та X_k описується формулою

$$K = sgn\left[\sum_{j=1}^{n} x_i^j x_i^k\right].$$

Для всіх m образів що запам’ятовано отримуємо формулу

$$K = \sum_{j=1}^{m} \sum_{k=1}^{m} K_{jk}.$$

Підвищення якості роботи мережі Хопфілда можна досягти завдяки використанню так званого «*алгоритму забування*». Суть цього підходу у тому, що у етапі формування матриці вагів поруч із «істинними» образами мережа запам'ятовує кілька «хибних» образів. Потім мережа отримує деяке вхідне значення і поступово досягає точки тяжіння, що відповідає деякому образу. Якщо цей образ X_F виявився хибним, то ваги мережі коригуються за формулою

$$W(t+1) = W(t) - \lambda X_F^T X_F.$$

Багаторазове повторення цієї процедури дозволяє виправити області тяжіння образів що запам’ятовано.

Для опису роботи мережі Хопфілда часто використовується поняття *енергетичної функції*, яку можна вибрати для пари нейронів у такому вигляді:

$$e_{ij} = x_i w_{ij} x_j.$$

Тоді енергія всієї мережі може бути описана формулою

$$E = -\sum_i \sum_j x_i w_{ij} x_j = -X^T W X.$$

Кожному образу, що у асоціативної пам'яті, відповідає свій мінімум енергії E.

Для виконання умови $E \to min$ потрібно, щоб

$$\sum_i \sum_j x_i w_{ij} x_j \to max.$$

Виходи мережі $Y \in \{-1, 1\}$, тому при виборі $w_{ij} = x_i x_j$ всі доданки стають позитивними, і отримуємо мінімум енергії

$$E = -\sum_i \sum_j (x_i)^2 (x_j)^2.$$

Можна також помітити, що за такого вибору ваг забезпечується

$$\sum_{j=1}^{N} w_{ij}x_j = \frac{1}{N}\sum_{j=1}^{N} x_i(x_j x_j) = x_i \frac{1}{N}\sum_{j=1}^{N} 1 = x_i.$$

Можна показати, що енергія мережі Хопфілда при симетричній матриці ваг або зменшується, або не змінюється.

Розглянемо енергію на момент часу *t*, виділивши у ній складову, внесену нейроном x_p:

$$E(t) = -\sum_i \sum_j x_i w_{ij} x_j = \sum_{i\neq p}\sum_{j\neq p} x_i w_{ij} x_j - \sum_j x_j w_{pj} x_p - \sum_i x_i w_{ip} x_p.$$

Нехай значення x_p у час $t + 1$ змінюється:

$$E(t+1) = -\sum_i \sum_j x_i w_{ij} x_j = \sum_{i\neq p}\sum_{j\neq p} x_i w_{ij} x_j - \sum_j x_j w_{pj} x_p^* - \sum_i x_i w_{ip} x_p^*.$$

Тоді

$$\Delta E = E(t+1) - E(t) = \\ = -\sum_j x_j w_{pj} x_p^* - \sum_i x_i w_{ip} x_p^* + \sum_j x_j w_{pj} x_p + \sum_i x_i w_{ip} x_p.$$

Оскільки вагова матриці симетрична, можливо записати:

$$\Delta E = 2\sum_i x_j w_{pj}(x_p - x_p^*).$$

Можливі два варіанти зміни енергії:

1. $x_p = -1, x_p^* = 1 \Rightarrow (x_p - x_p^*) = -2, \sum_i x_i w_{pj} > 0 \Rightarrow \Delta E < 0$

2. $x_p = 1, x_p^* = -1 \Rightarrow (x_p - x_p^*) = 2, \sum_i x_i w_{pj} < 0 \Rightarrow \Delta E < 0$

Таким чином, збільшення енергії завжди виявляється негативним.

За існуючими оцінками, інформаційна ємність мережі Хопфілда порівняно невелика – кількість випадкових образів *M*, які може запам'ятати мережа, пов'язана з числом нейронів мережі *N* формулою

$$M \leq 0{,}15N.$$

Розглянемо приклади роботи з мережею Хопфілда у MatLab.

Мережа Хопфілда має один нейронний шар з функціями зважування dotprod, накопичення netum і лінійною обмеженою функцією активізації satlins. Шар охоплений динамічним зворотним зв'язком з вагами *LW*{1,1} і має усунення зображені на рисунку 6.10.

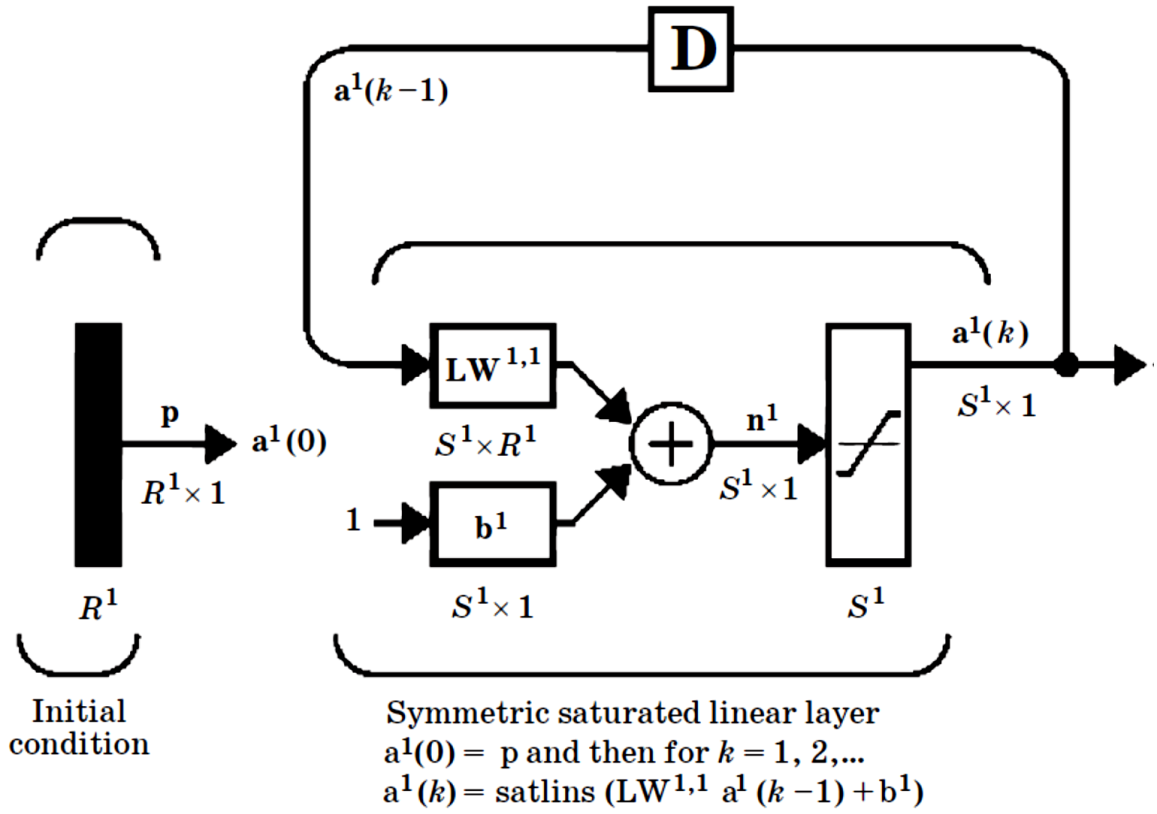


Рисунок 6.10 - Мережа Хопфілда в MatLab

Активаційна функція нейронів наведено на рисунку 6.11.

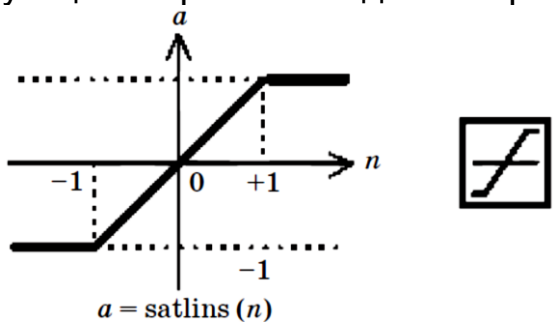


Рисунок 6.11 - Активаційна функція мережі Хопфілда

Функція створення мережі Хопфілда має вигляд

```
>> net = newhop(T)
```

де *T* - масив елементів розміром *R*×*Q*, що поєднує *Q* цільових векторів зі значеннями +1 або -1; *R* – кількість елементів вектору входу.

Приклад 6.4. Створити мережу Хопфілда з двома стійкими точками у тривимірному просторі:

```
>> T = [-1 -1 1; 1 -1 1] % визначення двох атракторів
T =
    -1     1
```

```
    -1    -1
     1     1

>> net = newhop(T); % створення мережі Хопфілда
>> A = T; % перевірка
>> Y = sim(net,2,[], A) % роботи мережі для еталонних образів
%
Y =
    -1     1
    -1    -1
     1     1
>> A = {[-0.6; -0.6; 0.6]}
A =
    {3×1 double}

>> [Y,Pf,Af] = sim(net,{1 5},{},A);
```

Можна подивитися, як змінювався вихід мережі після кожної ітерації:

```
>> [Y{1} Y{2} Y{3} Y{4} Y{5}]

ans =

   -0.6971   -0.8099   -0.9410   -1.0000   -1.0000
   -0.9884   -1.0000   -1.0000   -1.0000   -1.0000
    0.9884    1.0000    1.0000    1.0000    1.0000
```

Помилка поступово зменшується, і мережа переходить на стійкий стан.

Приклад 6.5. Розглянемо чотири атрактори у багатовимірному просторі:

```
vectors = [-1 1 -1 -1 1 -1 -1 1 -1; -1 -1 -1 1 1 1 -1 -1 -1;
    -1 -1 1 -1 1 -1 1 -1 -1; 1 -1 -1 -1 1 -1 -1 -1 1]';
 net = newhop(vectors);
 result = sim(net,4,[],vectors)
 test = {[0.1; 0.8; -1; -0.7; 0.5; -1; -0.9; 0.85; -1]};
 result = sim(net,{1,5},{},test);
 for i = 1:5,
disp(sprintf('Network state after %d iterations:',i));
disp(result{i});
end

result =
    -1    -1    -1     1
     1    -1    -1    -1
```

```
    -1    -1     1    -1
    -1     1    -1    -1
     1     1     1     1
    -1     1    -1    -1
    -1    -1     1    -1
     1    -1    -1    -1
    -1    -1    -1     1
```

Network state after 1 iterations:

```
   -0.4930
    0.8601
   -1.0000
   -1.0000
    0.9661
   -1.0000
   -1.0000
    0.8712
   -0.7384
```

Network state after 2 iterations:

```
   -0.7045
    0.9879
   -1.0000
   -1.0000
    1.0000
   -1.0000
   -1.0000
    0.9904
   -0.7593
```

Network state after 3 iterations:

```
   -0.8625
    1.0000
   -1.0000
   -1.0000
    1.0000
   -1.0000
   -1.0000
    1.0000
   -0.8748
```

Network state after 4 iterations:

```
   -0.9966
    1.0000
   -1.0000
```

```
-1.0000
 1.0000
-1.0000
-1.0000
 1.0000
-0.9993

Network state after 5 iterations:
   -1
    1
   -1
   -1
    1
   -1
   -1
    1
   -1
```

Приклад 6.6. Нехай точки тяжіння мережі Хопфілда задані на площині в кутах прямокутника (див. рис. 6.12):

```
T = [+1 -1; -1 +1; +1 +1; -1 -1];
T = T';
plot(T(1,:),T(2,:),'ro','MarkerSize',13), hold on;
axis ([-1.2 1.2 -1.2 1.2]);
title('Простір стану мережі Хопфілда');
xlabel('x');
ylabel('y');
net = newhop(T);
%Розглянемо процес зміни станів мережі Хопфілда для кількох
випадкових точок:
for i = 1:5
a = {rands(2,1)};
[y, Pf, Af] = sim (net, {1 20}, {}, a);
record = [cell2mat(a) cell2mat(y)];
start = cell2mat(a);
plot(start(1,1),start(2,1),'kx',record(1,:),record(2,:), '-','LineWidth',1)
end
```

Як випливає з рис. 6.13 з кожного випадкового стану (позначено хрестиком) мережа Хопфілда потрапляє в найближче положення рівноваги.

Приклад 6.7. Розглянемо задачу розпізнавання зображень цифр за допомогою ШНМ Хопфілда.

Закодуємо цифри так:

```
zero = [-1 -1 -1 -1 -1 -1 -1,
          -1 +1 +1 +1 +1 +1 -1,
```

-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 -1 -1];

Рисунок 6.12 - Координати точок тяжіння мережі Хопфілда

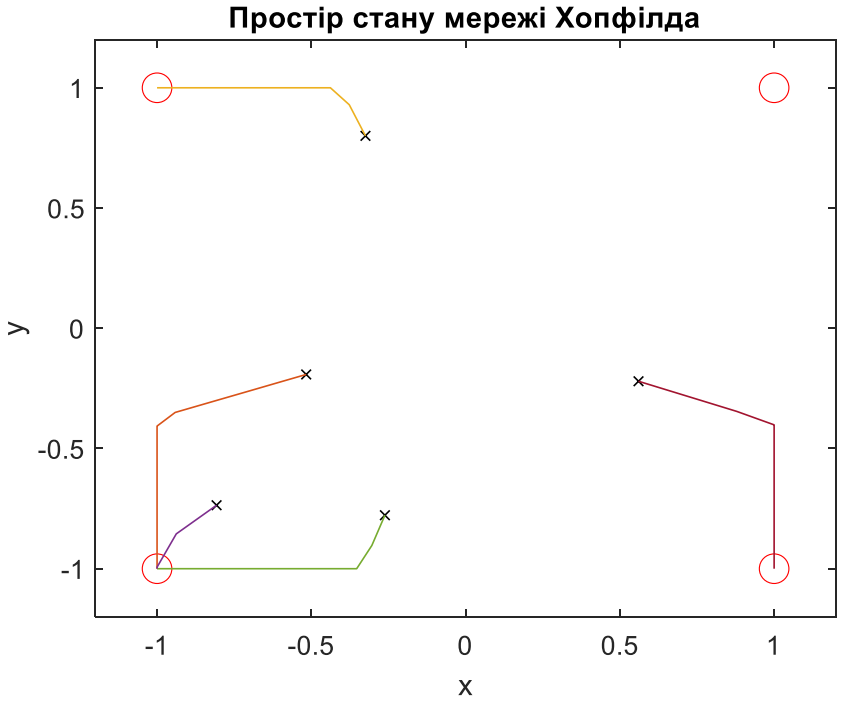


Рисунок 6.13 – Зміна станів мережі Хопфілда

```
zero = reshape (zero', 1,63); % перетворення у вектор-стовпець
one = [-1 -1 -1 -1 -1 -1 -1,
        -1 -1 -1 +1 -1 -1 -1,
        -1 -1 +1 +1 -1 -1 -1,
        -1 -1 -1 +1 -1 -1 -1,
        -1 -1 -1 +1 -1 -1 -1,
        -1 -1 -1 +1 -1 -1 -1,
        -1 -1 -1 +1 -1 -1 -1,
        -1 +1 +1 +1 +1 +1 -1,
        -1 -1 -1 -1 -1 -1 -1];
one = reshape(one',1,63);
two = [ -1 -1 -1 -1 -1 -1 -1,
        -1 +1 +1 +1 +1 +1 -1,
        -1 -1 -1 -1 -1 +1 -1,
        -1 -1 -1 -1 -1 +1 -1,
        -1 +1 +1 +1 +1 +1 -1,
        -1 +1 -1 -1 -1 -1 -1,
        -1 +1 -1 -1 -1 -1 -1,
        -1 +1 +1 +1 +1 +1 -1,
        -1 -1 -1 -1 -1 -1 -1];
two = reshape(two',1,63);
three = [ -1 -1 -1 -1 -1 -1 -1,
        -1 +1 +1 +1 +1 +1 -1,
        -1 -1 -1 -1 -1 +1 -1,
        -1 -1 -1 -1 -1 +1 -1,
        -1 -1 -1 +1 +1 +1 -1,
        -1 -1 -1 -1 -1 +1 -1,
        -1 -1 -1 -1 -1 +1 -1,
        -1 +1 +1 +1 +1 +1 -1,
        -1 -1 -1 -1 -1 -1 -1];
three = reshape(three',1,63);
four = [-1 -1 -1 -1 -1 -1 -1,
        -1 +1 -1 -1 -1 -1 -1,
        -1 +1 -1 -1 -1 -1 -1,
        -1 +1 -1 +1 -1 -1 -1,
        -1 +1 +1 +1 +1 -1 -1,
        -1 -1 -1 +1 -1 -1 -1,
        -1 -1 -1 +1 -1 -1 -1,
        -1 -1 -1 +1 -1 -1 -1,
        -1 -1 -1 -1 -1 -1 -1];
four = reshape(four',1,63);
five = [-1 -1 -1 -1 -1 -1 -1,
        -1 +1 +1 +1 +1 +1 -1,
        -1 +1 -1 -1 -1 -1 -1,
        -1 +1 -1 -1 -1 -1 -1,
```

```
            -1 +1 +1 +1 +1 +1 -1,
            -1 -1 -1 -1 -1 +1 -1,
            -1 -1 -1 -1 -1 +1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 -1 -1 -1 -1 -1 -1];
five = reshape(five',1,63);
six = [-1 -1 -1 -1 -1 -1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 +1 -1 -1 -1 -1 -1,
            -1 +1 -1 -1 -1 -1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 +1 -1 -1 -1 +1 -1,
            -1 +1 -1 -1 -1 +1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 -1 -1 -1 -1 -1 -1];
six = reshape(six',1,63);
seven = [-1 -1 -1 -1 -1 -1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 -1 -1 -1 -1 +1 -1,
            -1 -1 -1 -1 +1 -1 -1,
            -1 -1 -1 +1 -1 -1 -1,
            -1 -1 +1 -1 -1 -1 -1,
            -1 +1 -1 -1 -1 -1 -1,
            -1 +1 -1 -1 -1 -1 -1,
            -1 -1 -1 -1 -1 -1 -1];
seven = reshape(seven',1,63);
eight = [-1 -1 -1 -1 -1 -1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 +1 -1 -1 -1 +1 -1,
            -1 +1 -1 -1 -1 +1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 +1 -1 -1 -1 +1 -1,
            -1 +1 -1 -1 -1 +1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 -1 -1 -1 -1 -1 -1];
eight = reshape(eight',1,63);
nine = [-1 -1 -1 -1 -1 -1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 +1 -1 -1 -1 +1 -1,
            -1 +1 -1 -1 -1 +1 -1,
            -1 +1 +1 +1 +1 +1 -1,
            -1 -1 -1 -1 -1 +1 -1,
            -1 -1 -1 -1 -1 +1 -1,
            -1 -1 -1 -1 -1 +1 -1,
            -1 -1 -1 -1 -1 -1 -1];
```

```
nine = reshape (nine', 1,63);
% Задамо масив атракторів мережі Хопфілда (матриця 63×10):
digits1 = {zero, one, two, three, four, five, six, seven, eight, nine};
% Створимо мережу Хопфілда:
net = newhop(digits1);
% Для візуального представлення цифр можна використовувати такі
команди:
digits = {zero, one, two, three, four, five, six, seven, eight, nine};
bnw = [1 1 1; 0 1 0]; % колірна палітра
for P = 1:10 % виведення цифр на екран
      subplot(3,4,P);
      digit = digits {P};
      img = reshape (digit, 7,9);
      image((img'+1)*255/2);
      axis image
      axis off
      colormap(bnw)
      title(sprintf('Number %d', P));
end
```

Результат використання функції show наведено на рис. 6.14.

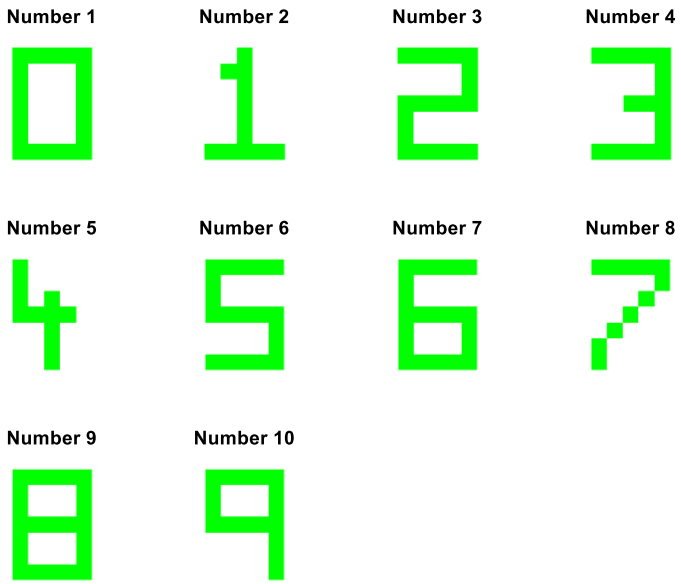


Рисунок 6.14 – Об'єкти, що запам'ятовано мережею Хопфілда

Далі можна перевірити роботу створеної мережі. Для цього опишемо спотворене зображення. Наприклад,

```
five1 = [-1 -1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 +1 -1 -1 -1 -1 -1,
```

```
+1 +1 -1 -1 -1 -1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 -1 -1 +1 +1 +1 -1,
-1 -1 -1 -1 -1 +1 -1,
-1 +1 +1 +1 +1 +1 -1,
-1 +1 -1 -1 -1 -1 -1];
five1 = reshape(five1',1,63);
% Відобразимо цей зашумлений код за допомогою команд
img = reshape(five1,7,9);
image((img'+1)*255/2);
axis image
axis off
colormap(bnw)
title(sprintf('Number 5 noise'))
```

Результат наведено на рис. 6.15.

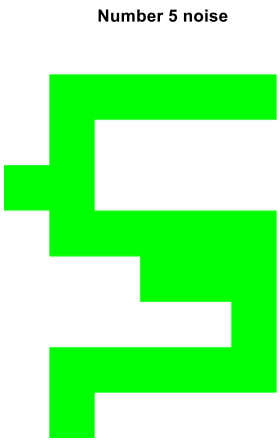


Рисунок 6.15 - Спотворений вхідний образ для мережі Хопфілда

Далі запустимо мережу Хопфілда:

```
five2 = {five1'}; % перетворення вхідних даних на масив
[Y,Pf,Af] = sim(net,{1 10},{},five2); % запуск мережі Хопфілда
% після чого зробимо виведення зображення на екран:
figure(3);
img = reshape (Y {10}, 7,9);
image((img'+1)*255/2);
axis image
axis off
colormap(bnw)
title(sprintf('Number 5 correct'));
```

Результат наведено на рис. 6.16.

Рисунок 6.16 - Зображення, відновлене мережею Хопфілда

6.3. Двонаправлена асоціативна пам'ять

Запропонований Хопфілдом підхід до організації асоціативної пам'яті є автоасоціативним, тобто можна відновити неповний образ або виправити спотворений, але не можна викликати якийсь інший, оскільки мережа має лише один шар.

Якщо використовувати більше шарів, то можна отримати гетероасоціативну пам'ять, в якій розмірність вихідного вектора може бути більшою за розмірність вхідного вектора.

Прикладом гетероасоціативної пам'яті є двонаправлена асоціативна пам'ять (ДАП).

Як і мережа Хопфілда, ДАП дозволяє виробляти правильні реакцію спотворені входи. Подібна мережа містить два шари, тому така пам'ять може мати вхідний вектор та вектор образу різної довжини (рис. 6.17).

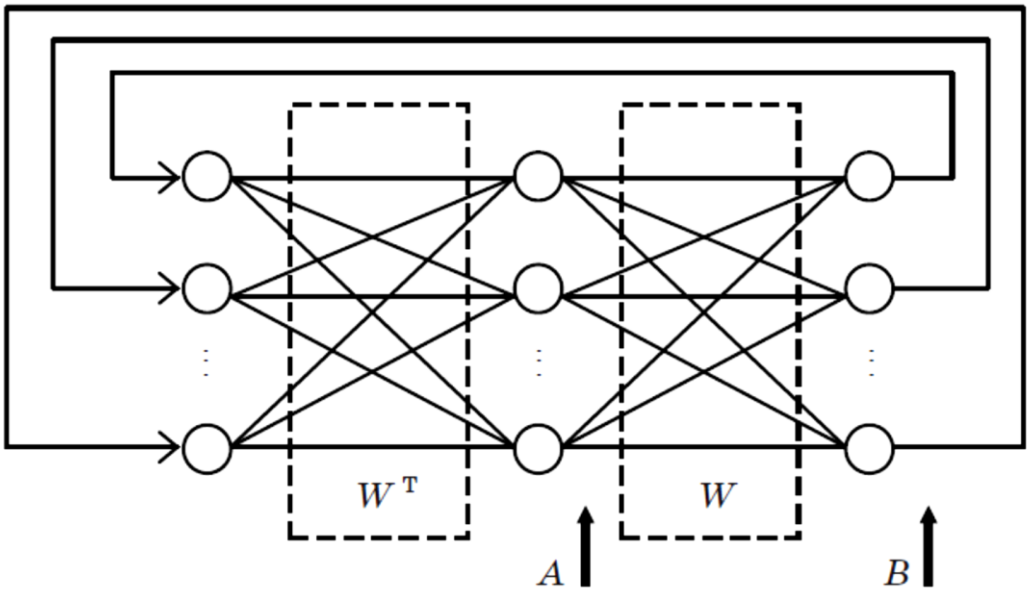


Рисунок 6.17 - Структура двонаправленої асоціативної пам'яті

Вхідний вектор A короткочасно виставляється на виході 1-го шару, так що мережа виробляє вихідний вектор:

$$B = F(AW),$$

де *F* – функція активації.

Потім вектор знімається, і мережа виробляє нове значення вхідного шару:

$$A = F(BW^T).$$

Кожен цикл може уточнювати вектори на виході 1-го та 2-го шару, доки не буде досягнуто резонансу і вектори *A* та *B* не стануть постійними. Вектор є асоційованим чином.

Вектори на виході шарів можна назвати *короткочасною пам'яттю*. Якщо прикласти на виході шару новий вхідний вектор, то стан короткочасної пам'яті може змінитися.

Система функціонує у бік мінімізації функції енергії, і кожен мінімум цієї функції відповідає одному запам'ятованому образу.

Довготривала пам'ять реалізується у вагах W і W^T.

Активаційна функція *F* може бути граничною з порогом 0.

Поведінка мережі змінюється за тактами. У проміжках між тактами вихід нейрона фіксовано.

Вектори *A* та *B* задають навчальний набір, і вагова матриця обчислюється за формулою

$$W = \sum_i A_i^T B_{i.}$$

Приклад 6.8. Нехай завдано навчальні пари:

A_1 = (100), B_1 = (0010),
A_2 = (0 1 0), B_2 = (1 0 0 1),
A_3 = (001), B_3 = (0100).

Нульове значення кодується за допомогою значення –1 (це покращує поведінку мережі):

A_1 = (1 -1 -1), B_1 = (-1 -1 1 -1),
A_2 = (-1 1 -1), B_2 = (1 -1 -1 1),
A_3 = (-1 -1 1), B_3 = (-1 1 -1 -1).

Далі розраховується матриця ваг:

$$W = A_1^T B_1 + A_2^T B_2 + A_3^T B_3,$$

$$A_1^T B_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} [-1 \quad -1 \quad 1 \quad -1] = \begin{bmatrix} -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & -1 & 1 \end{bmatrix},$$

$$A_2^T B_2 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} [1 \quad -1 \quad -1 \quad 1] = \begin{bmatrix} -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix},$$

$$A_3^T B_3 = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} [-1 \quad 1 \quad -1 \quad -1] = \begin{bmatrix} 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 \end{bmatrix},$$

$$W = \begin{bmatrix} -1 & -1 & 3 & -1 \\ 3 & -1 & -1 & 3 \\ -1 & 3 & -1 & -1 \end{bmatrix}, W^T = \begin{bmatrix} -1 & 3 & -1 \\ -1 & -1 & 3 \\ 3 & -1 & -1 \\ -1 & 3 & -1 \end{bmatrix}.$$

Нехай на вхід мережі поданий викривлений вектор *A* = (0,8 0,2 0). Тоді

$$B = F[AW] = F\left[[0{,}8 \quad 0{,}2 \quad 0] \begin{bmatrix} -1 & -1 & 3 & -1 \\ 3 & -1 & -1 & 3 \\ -1 & 3 & -1 & -1 \end{bmatrix}\right] =$$
$$= F[-0{,}2 \quad -1 \quad 2{,}2 \quad -1] = [0 \quad 0 \quad 1 \quad 0],$$

$$A = F[BW^T] = F\left[[-1 \quad -1 \quad 1 \quad -1] \begin{bmatrix} -1 & 3 & -1 \\ -1 & -1 & 3 \\ 3 & -1 & -1 \\ -1 & 3 & -1 \end{bmatrix}\right] =$$
$$= F[6 \quad -6 \quad -2] = [1 \quad 0 \quad 0],$$

Отже, використовуючи ДАП, можна пов'язувати одні образи з іншими, які мають іншу розмірність. Крім того, можна організовувати ланцюжки асоціацій, ніби відтворюючи образ за образом.

Важлива властивість ДАП полягає у її стійкості, яка зберігається за будь-яких значень ваг мережі.

Описаний варіант ДАП є найпростішим. У загальному випадку поведінка нейронів може бути асинхронною, функція активації може бути сигмоїдною і відповідно вихід нейронів може бути безперервним, а не дискретним. Це, наприклад, при оптичної реалізації ДАП. Тому в загальному випадку ваги ДАП не призначаються шляхом обчислення (як показано раніше), а виходять у результаті навчання.

Розглянемо, як можна використовувати принцип машини

Больцмана під час навчання ДАП [25].

Спочатку вагам ДАП присвоюються випадкові значення. Вводяться штучна температура *T* мережі і штучна енергія нейрона *E*. Штучна температура отримує високе початкове значення.

Штучну енергію нейрона розраховують за формулою

$$E_j = Y_j - \theta$$

де θ - порогове значення.

Для кожного нейрона розраховується величина

$$P_j = \frac{1}{\left(1 + exp\left(-\frac{E_j}{T}\right)\right)}.$$

Стан нейрона знаходять шляхом порівняння P_j та випадкового числа

$k \in [0,1]$. Якщо $P_j > k$, то $Y_j = 1$.

Процедура навчання включає три циклічно повторювані етапи:

1 етап

1.1. На вхід та вихід мережі короткочасно подається навчальна пара векторів.

1.2. У мережі відбувається динамічний процес до досягнення рівноваги.

1.3. Записуються вихідні значення всіх нейронів.

1.4. Повторюються кроки 1.1 – 1.3 всім повчальних пар.

1.5. Обчислюється так звана закріплена ймовірність P_{ij}^{+} як відношення числа дослідів, коли виходи нейронів *i* та *j* одночасно дорівнюють одиниці, до загального числа дослідів.

2 етап

2.1. Мережі надається випадковий стан та розглядається стан після завершення перехідного процесу. Виходи нейронів фіксуються.

2.2. Крок 2.1 повторюється багато разів.

2.3. Обчислюються так звані незакріплені ймовірності P_{ij}^{-} як відношення числа дослідів, коли виходи нейронів *i* та *j* одночасно дорівнюють одиниці, до загального числа дослідів.

3 етап

3.1. Ваги мережі коригуються за формулою

$$w_{ij} = w_{ij} + \eta\left(P_{ij}^{+} - P_{ij}^{-}\right),$$

де η - константа навчання.

6.4. Нейрона мережа Хеммінга

Нейронна мережа Хеммінга є моделлю асоціативної пам'яті. Її можна розглядати як розвиток мережі Хопфілда.

Основна ідея функціонування цієї мережі полягає в паралельному обчисленні відстані Хеммінга між вхідним вектором, що пред'являється на входи мережі, і образами, закодованими в структурі мережі. При цьому завдання формування еталонного образу на виході мережі не ставиться, достатньо видавати його номер.

Відстанню Хеммінга між двома двійковими векторами називається число компонент, у яких ці вектори різні.

У мережі Хеммінга, як і в мережі Хопфілда, *вхідні сигнали* $X = [x_1, x_2, \ldots, x_n]^T$ *можуть бути лише двійковими векторами*, що подаються за допомогою біполярного кодування: $x_i = +1$ (двійкова «1»), $x_i = -1$ (Двійковий «0»).

Ідея роботи мережі Хеммінга полягає у знаходженні відстані від вхідного двійкового вектору довжиною N до всіх образів, що зберігаються в мережі L. Таким чином, розмір входу та виходу мережі в загальному випадку не збігається. Після закінчення перехідного процесу в мережі залишається збудженим один із нейронів вихідного шару.

Найпростіший варіант ШНС Хеммінга реалізується у вигляді одношарової структури (див. рис. 6.18).

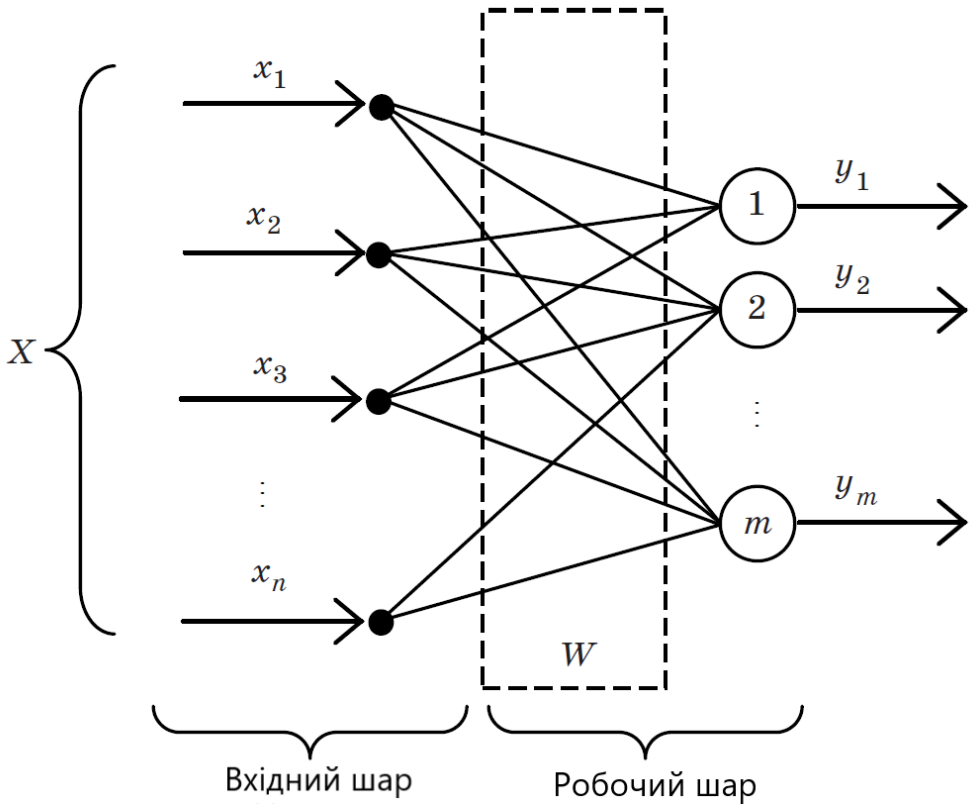


Рисунок 6.18 - Структура одношарової ШНС Хеммінга

Робочий шар мережі містить *m* нейронів (за кількістю закодованих у мережі образів) та обчислює відстані Хеммінга $y_1, y_2, \ldots, y_m$ між вхідним вектором *X* та кожним чином.

Розглянемо два біполярних вектору $X = [x_1,\ x_2, \cdots, x_n]$ та $W = [w_1,\ w_2, \cdots, w_n]$. Їх скалярний твір можна описати так:

$$XW^T = \sum_{i=1}^{n} x_i w_i = a - b,$$

де *a* - число однакових компонентів векторів; *b* – кількість різних його компонентів.

Наприклад:

$$n = 5,\ W = [1, -1, 1, -1, -1], \qquad X = [-1, -1, -1, 1, 1]$$

Тоді

$$XW^T = 1 \cdot (-1) + (-1) \cdot (-1) + 1 \cdot (-1) + (-1) \cdot 1 + (-1) \cdot 1 = 1 - 4 = -3.$$

Оскільки *n=a+b*, можна записати

$$XW^T = a - b = a - (n - a) = 2a - n.$$

Відповідно

$$a = \frac{n}{2} + \frac{XW^T}{2} = \frac{n}{2} + \frac{1}{2}\sum_{i=1}^{n} x_i w_i.$$

Цією формулою можна описати роботу нейронів вихідного шару.

Матриця ваг *W* робочого шару мережі формується на основі пред'явлених навчальних даних:

$$w_{ij} = -x/2\,, \quad i = \overline{1, n}, \quad j = i = \overline{1, m},$$

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix} = \begin{bmatrix} -x_{11}/2 & -x_{12}/2 & \cdots & -x_{1m}/2 \\ -x_{21}/2 & -x_{22}/2 & \cdots & -x_{2m}/2 \\ \vdots & \vdots & \vdots & \vdots \\ -x_{n1}/2 & -x_{n2}/2 & \cdots & -x_{nm}/2 \end{bmatrix}.$$

Нейрони робочого шару проводять обчислення за звичайною формулою

$$y_i = F\left(\sum_{i=1}^{n} x_i w_i - P\right), \quad i = \overline{1,n}, \quad j = i = \overline{1,m},$$

де *P* – поріг нейрона; *F* – його активаційна функція.

Для того, щоб нейрони робочого шару могли реалізувати обчислення відстані Хеммінга між пред'явленим вектором *X* і кожним з m образів, для них встановлюється поріг *P* = -*n*/2, а активаційна функція F вибирається як лінійна з насиченням:

$$F(z) = \begin{cases} 0, & \text{якщо } z \le 0, \\ z_i, & \text{якщо } 0 < z \le n, \\ n, & \text{якщо } z > n. \end{cases}$$

Графік активаційної функції нейронів робочого шару має вигляд, наведений на рис. 6.19.

Наприклад,

$$n = 5, \; W = [0{,}5; 0{,}5; 0{,}5; -0{,}5; -0{,}5], \qquad X = [-1, -1, -1, 1, 1]$$

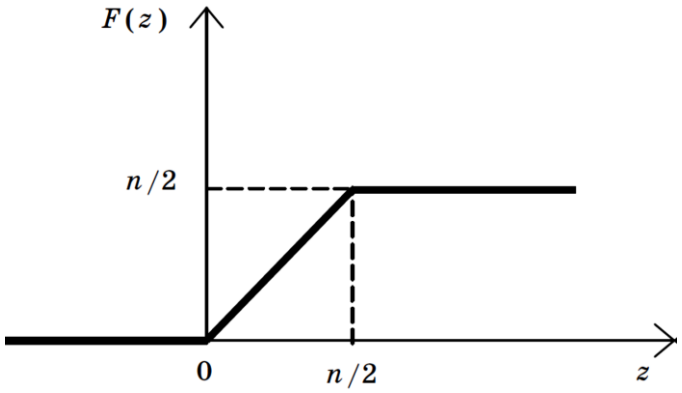


Рисунок 6.19 – Графік активаційної функції нейронів робочого шару

тоді

$$y = F(XW^T + 2{,}5) = F\big((-0.5) \cdot 3 + (-0.5) \cdot 2 + +2.5\big) = F(0) = 0.$$

Таким чином, вхідний і ваговий вектори не мають бітів, що збігаються (відстань Хеммінга максимально).

Якщо ж розглянути пару

$$n = 5, \; W = [0{,}5; 0{,}5; 0{,}5; -0{,}5; -0{,}5], \qquad X = [1, 1, 1, -1, -1]$$

тоді

$$y = F(XW^T + 2{,}5) = F\big((0.5)\cdot 3 + (0.5)\cdot 2 + +2.5\big) = F(5) = 5.$$

Тут відстань Хеммінга мінімальна, і вихідний сигнал є максимальним.

Другий шар у мережу Хеммінга додається з метою виділення образу, що *найбільше відповідає вхідному вектору*. Цей шар (часто званий *maxnet*) має зворотний зв'язок організації конкуренції нейронів (див. рис. 6.20).

Другий шар, як і перший містить *m* нейронів. Усі нейрони шару пов'язані між собою зворотними зв'язками за принципом «кожен з кожним». Разом з тим, на відміну від мережі Хопфілда, всі нейрони 2-го шару мають ненульові зворотні зв'язки на самих себе. Ваги всіх зв’язків постійні, причому різноіменні нейрони пов'язані негативним (гальмівним) зворотним зв'язком з вагою -*ε*, а на самих себе нейрони мають позитивний (збудливий) зворотний зв'язок з вагою, що дорівнює +1.

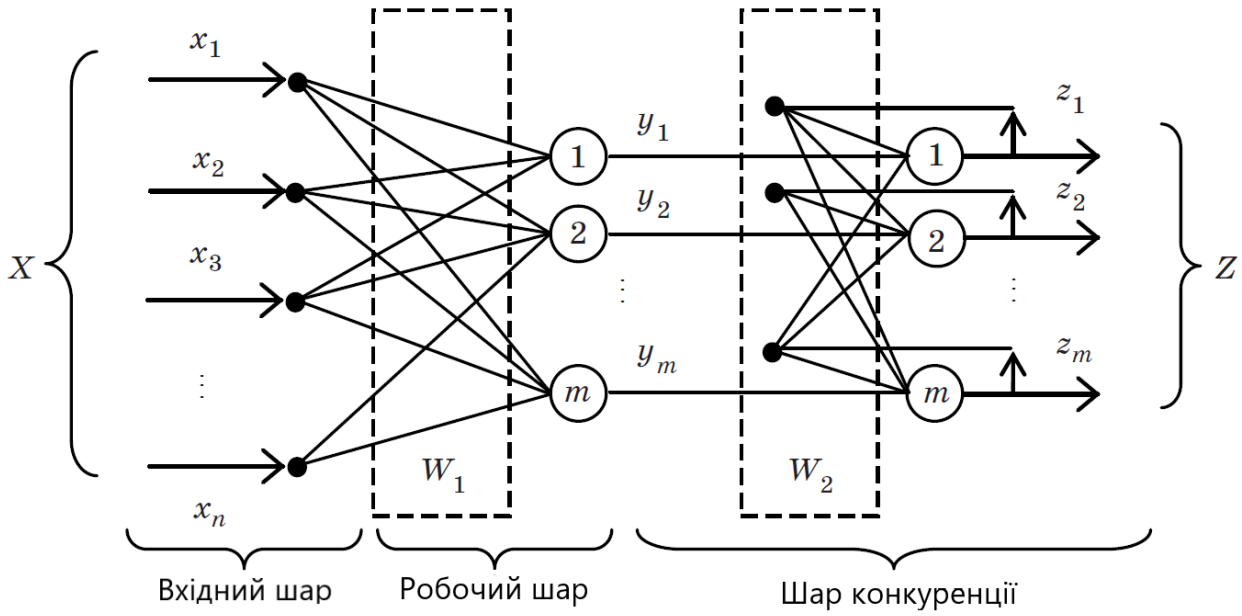


Рисунок 6.20 – Структура двошарової ШНС Хеммінга

Матриця ваг 2-го шару має вигляд

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix} = \begin{bmatrix} 1 & -\varepsilon & \cdots & -\varepsilon \\ -\varepsilon & 1 & \cdots & -\varepsilon \\ \vdots & \vdots & \vdots & \vdots \\ -\varepsilon & -\varepsilon & \cdots & 1 \end{bmatrix}.$$

Другий шар функціонує в режимі «*переможець отримує все*», тобто в режимі, що встановився, в шарі залишається активованим тільки один нейрон, а інші знаходяться в незбудженому стані. Відмінне

від нуля значення вихідного сигналу активованого нейрона з номером *k* вказує на належність вхідного образу відповідного класу.

Двошарова мережа Хеммінга має дві фази функціонування. У першій фазі входи мережі подається вхідний вектор *X*. На виходах 1-го робочого шару формуються вихідні сигнали, які задають початкові стану *Z*(0) нейронів 2-го шару maxnet.

У другій фазі сигнали, що ініціюють шар maxnet, видаляються, і зі сформованого ними початкового стану запускається ітераційний процес усередині 2 шару. Ітераційний процес завершується в момент, коли всі нейрони, крім одного, перейдуть у нульовий стан. Нейрон-переможець стає представником класу даних, до якого належить вхідний вектор *X*. Процес визначення нейрона-переможця виконується покрокова відповідно до виразу

$$z_k(t) = F\left(\sum_{j=1}^{m} w_{jk} z_j(t-1)\right), \quad j,k = \overline{1,m}.$$

Активаційна функція *F* тут – лінійна з насиченням, причому величина порога має бути досить великою, щоб будь-які можливі значення аргументу не призводили до насичення.

Враховуючи, що ваги 2-го шару вибираються виходячи з умови

$$w_{jk} = \begin{cases} 1, & j = k, \\ -\varepsilon, & j \neq k, \end{cases}$$

процес визначення нейрона-переможця можна подати у вигляді

$$z_k(t) = F\left(z_k(p-1) - \varepsilon \sum_{j \neq k}^{m} z_j(p-1)\right), \qquad k = \overline{1,m}.$$

Значення ваг гальмівних зв'язків ε зазвичай вибирається в діапазоні

$$0 < \varepsilon < \frac{1}{m-1}.$$

Для забезпечення абсолютної збіжності процесу у шарі maxnet ваги гальмівних зв'язків повинні відрізнятись один від одного. Для задоволення цієї умови при розрахунку ваг гальмівних зв'язків додається мінімальна випадкова величина

$$w_{jk} = -\frac{1}{L-1} + \xi.$$

Мережа Хеммінга може бути модифікована, щоб видавати на виході не номер класу, якого належить вхідний вектор, а асоційований із цим класом образ. При цьому довжина вихідного вектора може не збігатися з довжиною вектора вхідного (*гетероасоціативна пам'ять*). Для того, щоб досягти такого ефекту, до мережі Хеммінга додається 3-й (вихідний) шар (див. рис. 6.21).

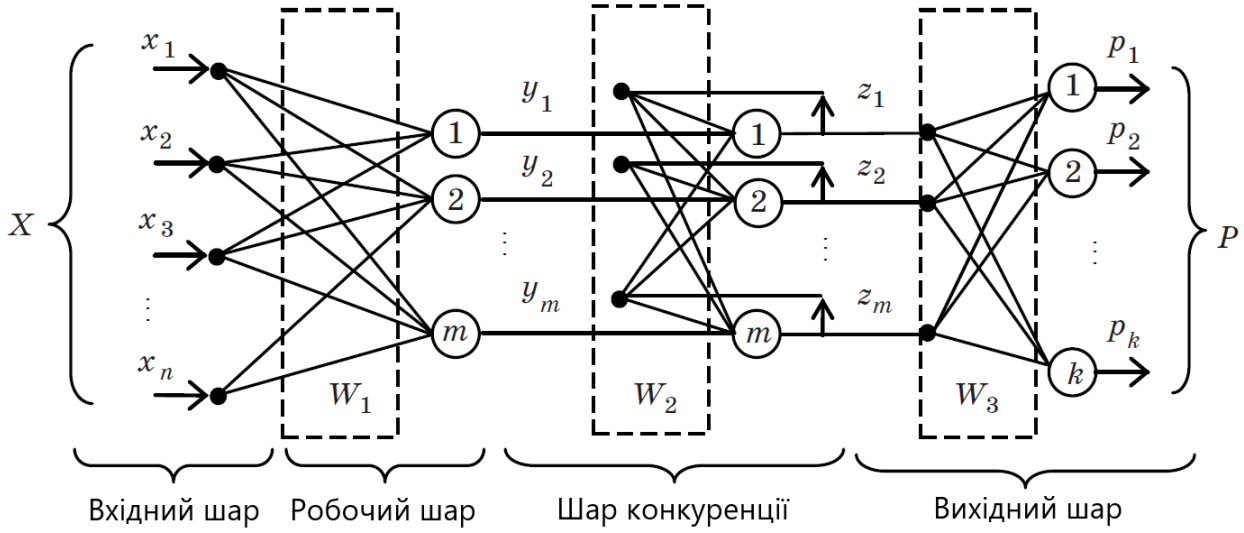


Рисунок 6.21 – Структура тришарової ШНМ Хеммінга

У процесі функціонування тришарової мережі Хеммінга на відміну двошарової з'являється третя фаза. У цій фазі нейрон-переможець у 2-му шарі за допомогою ваг, що зв'язують його з нейронами 3-го вихідного шару, формує на виході відгук мережі у вигляді двійкового вектора *P*, що відповідає збуджуючому вектору *X*. Оскільки на виході шару конкуренції тільки один нейрон видає одиничний вихідний сигнал, можна сформулювати просте правило призначення ваги вихідного шару:

$$w_{il} = p_{il}, \quad i,j = \overline{1,m}, \qquad l = \overline{1,m},$$

$$W_3 = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_m \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mk} \end{bmatrix}.$$

Мережа Хеммінга має важливі переваги в порівнянні з мережею Хопфілда:

1. Ємність даної мережі не залежить від розмірності вхідного

сигналу і дорівнює числу нейронів робочого шару.

2. Число міжнейронних зв'язків у мережі Хеммінга менше, ніж у мережі Хопфілда. Наприклад, мережа Хопфілда з розмірністю вхідного сигналу 100 може запам'ятати приблизно 13 випадкових образів, вона буде містити 10000 зв'язків між нейронами. Одношарова мережа Хеммінга з такою ж ємністю міститиме 1000 зважених зв'язків, а двошарова – 1100, з яких 100 зв'язків – у шарі maxnet.

3. Одношарова мережа Хеммінга працює істотно швидше за мережу Хопфілда, оскільки вирішення завдання формується в результаті одноразового проходу через один шар нейронів.

4. Мережа Хеммінга має один із найпростіших алгоритмів формування ваг та зміщень.

5. Експериментально доведено, що двошарова мережа Хеммінга функціонує краще, ніж мережа Хопфілда при випадковому наборі векторів, що запам'ятовуються.

Однак у мережі Хеммінга є й недоліки:

1. Погана робота при сильно зашумлених вхідних сигналах. Якщо сигнали знаходяться на однаковій відстані Хеммінга від двох або більше еталонів, то вибір одного з еталонів стає випадковим.

2. Мережі Хеммінга розраховані працювати лише з бінарними вхідними сигналами, що обмежує їх застосування.

Однак у цілому мережі Хеммінга можуть успішно використовуватися для вирішення завдань розпізнавання образів, класифікації, реалізації асоціативної та гетероасоціативної пам'яті, а також передачі сигналів в умовах перешкод.

6.5 Адаптивні резонансні нейронні мережі

Традиційні ШНС важко вирішують проблему стабільності та одночасної пластичності запам'ятовування, тобто проблему нарощування числа образів у пам'яті без спотворення тих, які там вже є.

Мережа прямого поширення навчається на фіксованій безлічі прикладів. Якщо після навчання кількість прикладів збільшується, то «донавчання» неможливо, потрібно повне перенавчання.

У мережах Кохонена також передбачається, що це вхідні образи належать до одного з отриманих під час навчання класів. Виявлення нового класу потребує повного повторення процесу навчання.

Аналогічна ситуація спостерігається під час використання мереж Хопфілда та Хеммінга.

Мережа АРТ призначена на вирішення проблеми стабільності – пластичності. Існують так звані АРТ1-мережі, що обробляють двійкові вектори, а також АРТ2-мережі для обробки безперервних векторів. Крім того, слід назвати моделі ARTMAP та FuzzyART.

Розглянемо роботу АРТ1.

Мережа АРТ1 є векторним класифікатором. Вхідний вектор відноситься до однієї із груп раніше запам'ятаних векторів. Рішення про класифікацію видається як порушення одного з нейронів шару розпізнавання.

Відмінність АРТ1 від інших ШНС полягає в наступному:

1) якщо вхідний вектор схожий на один із раніше запам'ятаних векторів, то запам'ятований вектор буде змінюватися, щоб стати схожим на вхідний вектор;

2) можлива відмова від класифікації, якщо вхідний вектор не схожий на жоден із запам'ятованих векторів. У цій ситуації утворюється новий клас.

Спрощена конфігурація мережі АРТ1 [25] включає шар порівняння, шар розпізнавання, скидання, приймач 1 і приймач 2 (див. рис. 6.22).

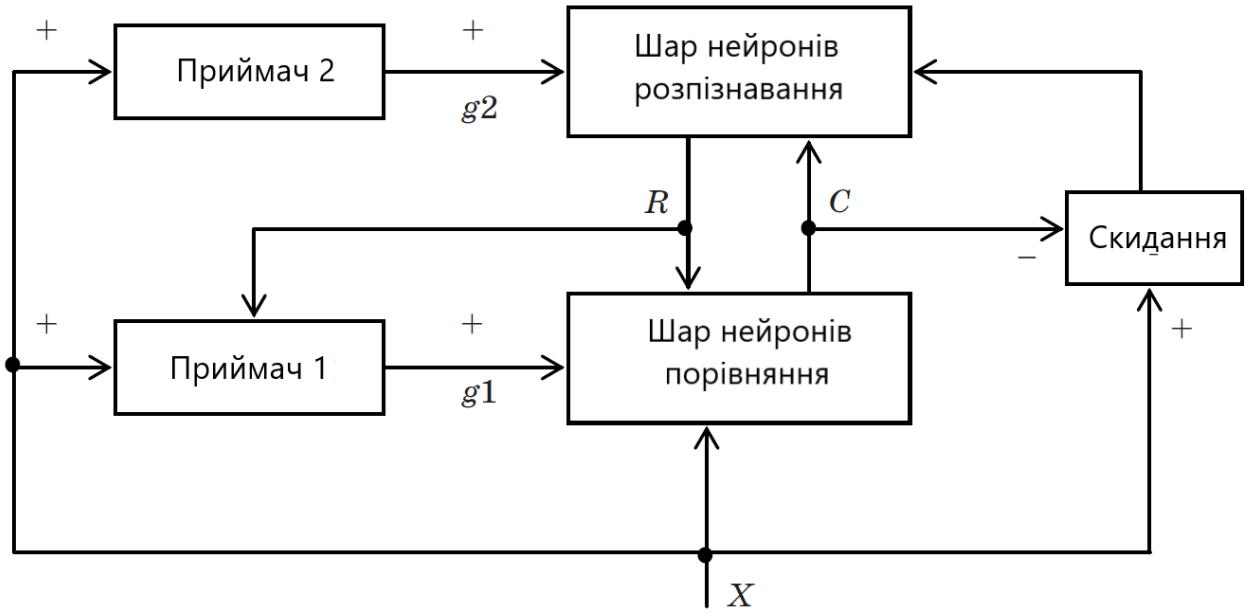


Рисунок 6.22 – Структура нейронної мережі АРТ

Введемо позначення: m – розмір вхідного вектора; n – кількість запам'ятаних образів.

Шар порівняння отримує двійковий вхідний вектор X і спочатку пропускає його незмінним для формування вихідного вектора C (на більш пізній стадії компоненти вектора R модифікують C).

Спрощено шар порівняння має вигляд, наведений на рис. 6.23.

Вектори T утворюються ваговими коефіцієнтами t_{ij}, які є двійковими числами.

Кожен нейрон шару порівняння ($ШН_1$ – $ШНm$) отримує три двійкові входи:

1) компонент вхідного вектора x_i;

2) сигнал зворотного зв'язку p_j;

3) вхід від приймача *g*1.

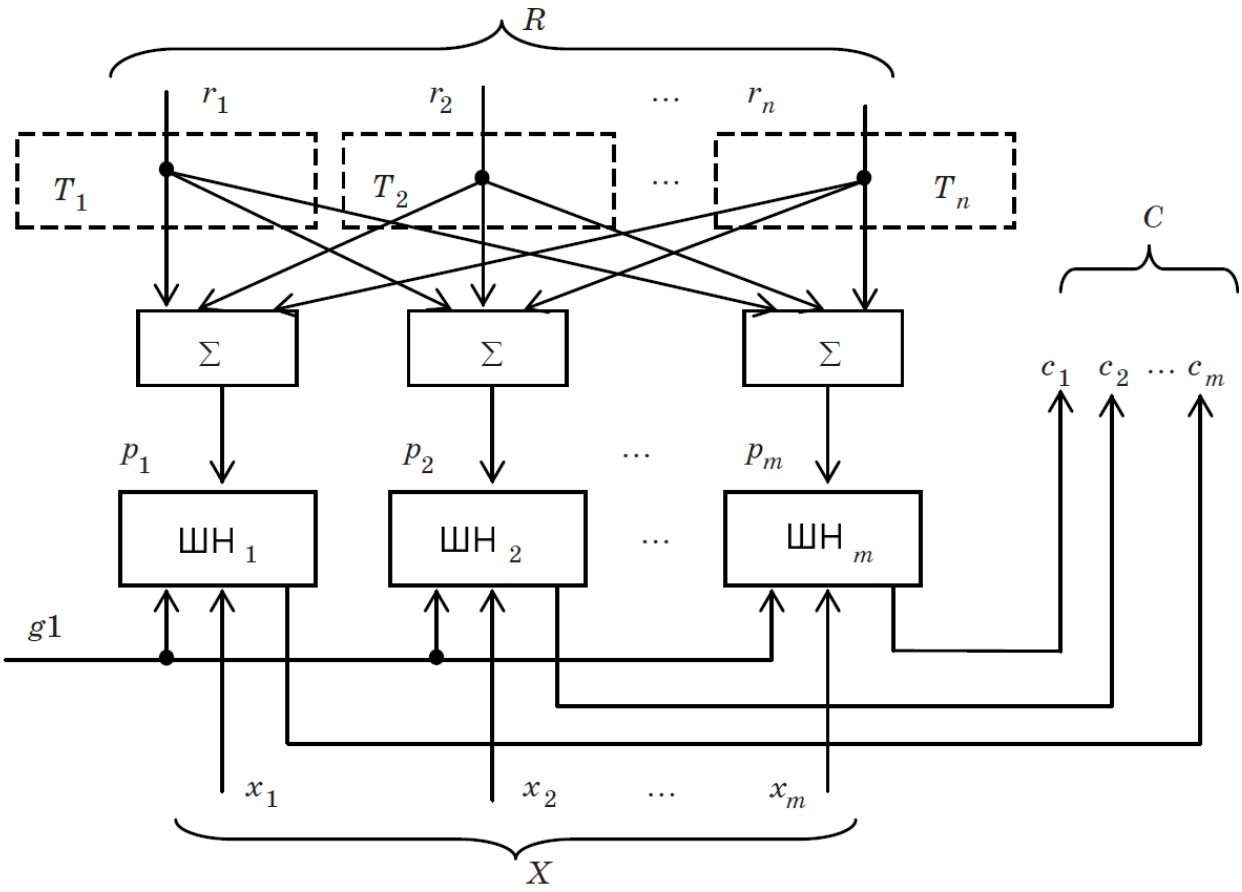


Рисунок 6.23 - Структура шару порівняння

Для того щоб на виході нейрона була одиниця, необхідно, щоб не менше двох його входів дорівнювали одиниці. Спочатку сигнал *g*1 встановлений в "1", а сигнали *p*j - в "0", тому вихідний вектор *C* збігається з *X*.

Шар розпізнавання є класифікацією вхідних векторів. Кожен нейрон шару розпізнавання має вектор ваги *B*. Тільки один нейрон у цьому шарі збуджується, всі інші загальмовані. Це досягається завдяки тому, що всі нейрони розпізнавання шару охоплені механізмом латерального збудження-гальмування (див. рис. 6.24).

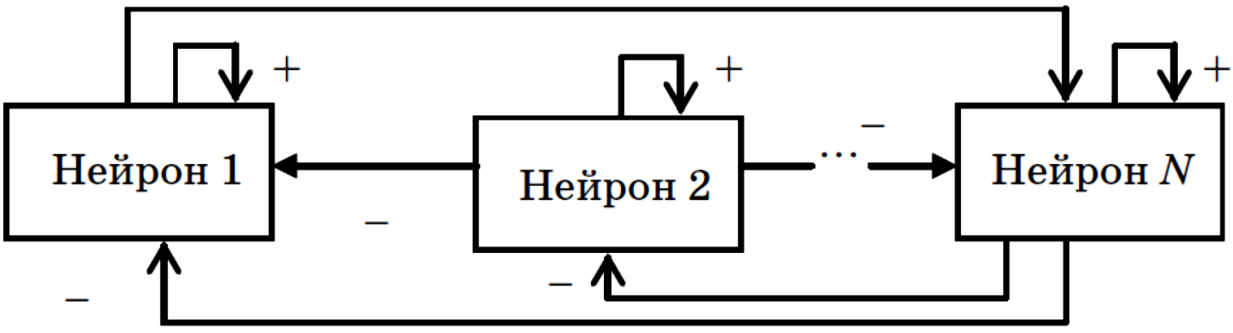


Рисунок 6.24 – Взаємодія нейронів шару розпізнавання

Таким чином, чим більше нейрон збуджений, тим більше він гальмує інші нейрони та одночасно підтримує свій рівень.

Спрощену версію шару розпізнавання наведено на рис. 6.25.

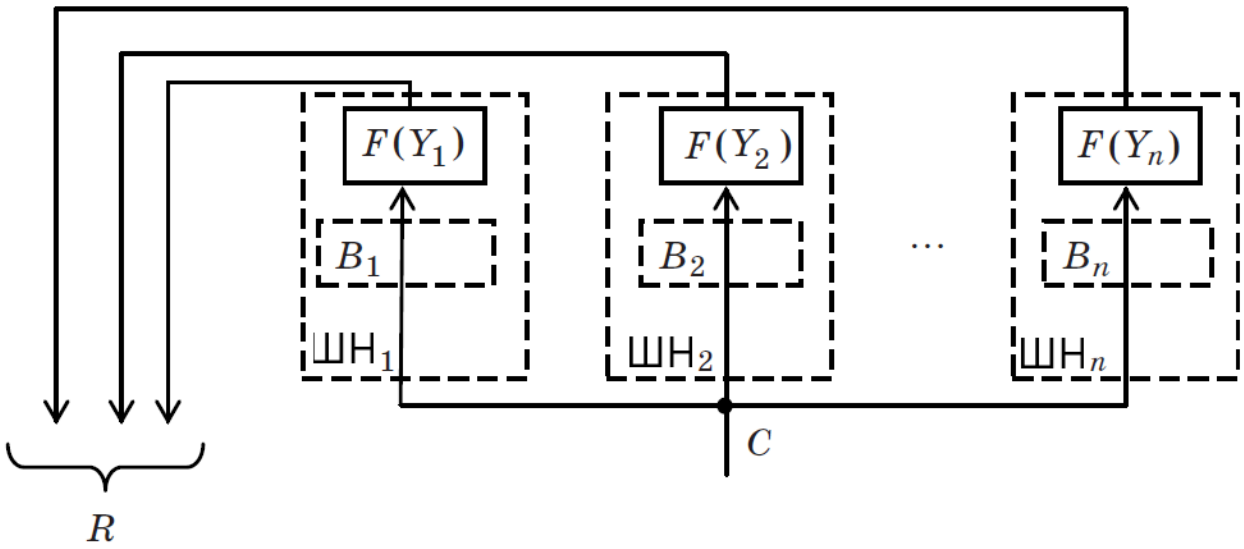


Рисунок 6.25 - Взаємодія нейронів шару розпізнавання

На входи всіх нейронів шару розпізнавання надходять сигнали від усіх нейронів шару порівняння, і навпаки.

Кожен нейрон шару розпізнавання має вектор ваги *B*. Вихід нейрона

$$Y_j = B_j C.$$

Активаційна функція *F* порогова:

$$F(Y_j) = \begin{cases} 1, & Y_j \geq T, \\ 0, & Y_j < T. \end{cases}$$

Таким чином, вектор *R* двійковий.

Нейрон *j* має максимальну реакцію, якщо вектор *C* (вихід шару порівняння) максимально схожий на його ваговий вектор *Bj*. Таким чином, ваги нейрона являють собою запам'ятований образ для групи векторів.

Ваги є дійсними числами. Двійкова версія цього образу запам'ятовується у наборі ваги шару порівняння.

У процесі функціонування кожен нейрон шару розпізнавання обчислює згортку вектора власних ваги і вхідного вектора *C*. Чим ближче ваги до вектора *C*, то вище вихід нейрона. Нейрон-переможець загальмовує інші нейрони шару розпізнавання.

Таким чином, лише один нейрон шару розпізнавання матиме на виході «1», а решта – «0». Отже, вектор R матиме лише один одиничний

компонент.

Приймач 2. Його вихід дорівнює одиниці, якщо вхідний вектор *X* має хоча б одну одиничну компоненту. Інакше кажучи, сигнал *g*2 є логічним АБО компонентом вхідного вектора:

$$g2 = (x_1 \vee x_2 \vee x_3 \vee \cdots \vee x_m)$$

Приймач 1. Працює так само, як приймач 2, але прихід хоча б однієї одиничної компоненти *R* скидає *g1* «0»:

$$g1 = (x_1 \vee x_2 \vee \cdots \vee x_m) \wedge \overline{(r_1 \vee r_2 \vee \cdots \vee r_m)}$$

Скидання. Модуль скидання вимірює подібність між векторами *X* і *C*. Мірою подібності служить відношення числа одиниць *X* до їх числа *C*:

$$S = N/D$$

де *D* - число одиниць у векторі з найбільшим числом одиниць.

Наприклад,

$$X = [1011101] \quad (D{=}5)$$
$$C = [0011101] \quad (N{=}4)$$

Таким чином параметр подібності змінюється від 0 до 1.

Три основні операції АРТ полягають у розпізнаванні, порівнянні та пошуку.

Розпізнавання полягає у наступному. Поки вхідний вектор *X* відсутній, всі його компоненти можна вважати нульовими, отже, *g2* = 0, і в «0» встановлюються виходи всіх нейронів шару розпізнавання (так як *C* = *X*).

Потім подається вхідний вектор *X*, тобто одна або більше компонентів вхідного вектора стають ненульовими, а *g1* та *g2* - рівними одиниці. Це створює умови для збудження нейронів шару порівняння, і вектор дублює вектор *X*.

Після цього у шарі розпізнавання обчислюється згортка вектора ваг B_j та *C*. Нейрон, ваги якого найбільш схожі на вхідний вектор, перемагає та загальмовує інші нейрони. Єдина компонента *R* стає одиничною. Таким чином, один нейрон шару розпізнавання відповідає одній з класифікаційних категорій.

Порівняння. Компоненти вектора *R* надходять на всі нейрони шару порівняння через ваги t_{ij}, які набувають двійкових значень.

Взаємозв'язок між векторами ваг *T* та *B* полягає в тому, що є масштабованою версією вектора *T*.

Оскільки *R* більше не нульовий, сигнал *g1* встановлюється «0»,

тому відповідно з правилом 2/3 збудитися можуть тільки нейрони, що отримують на вході одночасно одиниці від вхідного вектора *X* і вектора *R*.

Якщо *X* і *P* не мають компонент які збігаються, то зворотний зв'язок від шару, що розпізнає, скине компоненти *C* в «0» (див. рис. 6.26).

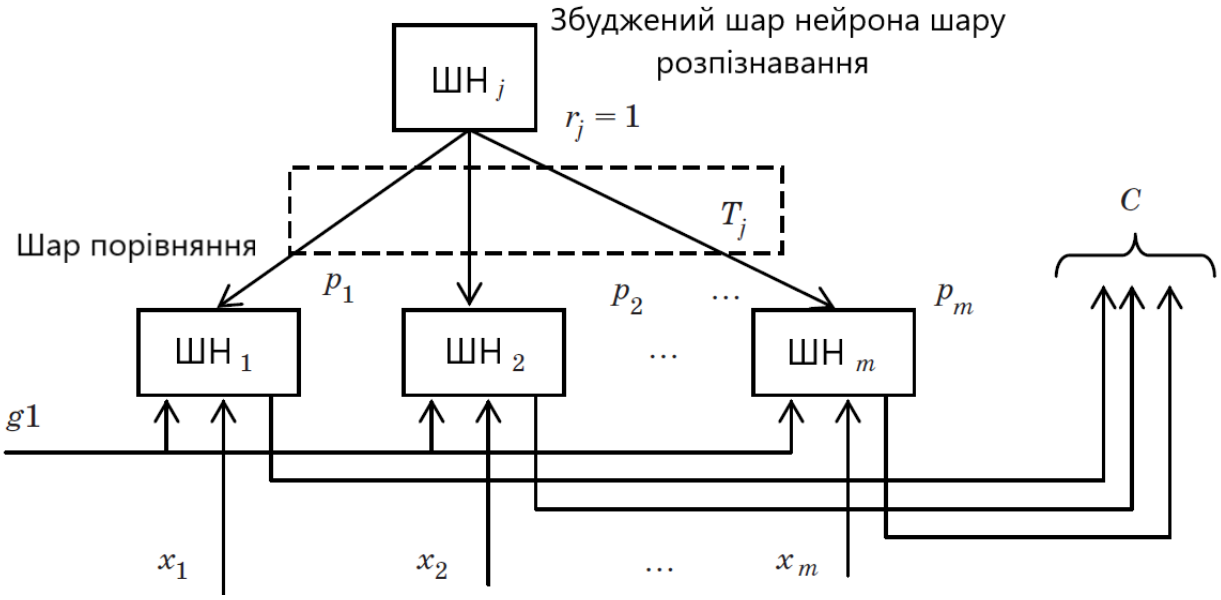


Рисунок 6.26 - Взаємодія шарів розпізнавання та порівняння

Якщо відмінностей між векторами *X* і *P* багато, то вектор буде мати нулів значно більше, ніж *X*, і виробляється сигнал скидання, який відключає збуджений нейрон на час поточної класифікації.

Пошук. Якщо сигнал скидання не виробляється, подібність вважається знайденим, а класифікація завершеною. Якщо ж скидання відбулося, то всі компоненти R обнуляються, g1 встановлюється в «1», вектор стає рівним X, і відбувається нова перевірка, але вже без загальмованого нейрона. Ця процедура продовжується доти, доки не будуть загальмовані всі нейрони. У цьому випадку відбувається запам'ятовування нового образу, для чого виділяється новий нейрон, вагові коефіцієнти якого B і T встановлюються відповідно до нового способу.

Якщо ж буде знайдено один з нейронів, схожий на вхідний образ, то проводиться навчальний цикл з метою корекції ваг *B* та *T*.

При навчанні на вхід мережі послідовно подаються вектори з навчальної множини і підлаштовуються ваги мережі для того, щоб подібні вектори активізували один і той же нейрон.

Можливі різні алгоритми навчання. Розглянемо так званий *швидкий алгоритм*.

Вага зв'язку b_{ij}, що зв'язує *i*-й нейрон шару порівняння та *j*-й нейрон шару розпізнавання, може бути знайдений за формулою

$$b_{ij} = \frac{2C_i}{\sum_k C_k + 1},$$

де j – збуджений нейрон; Σ - число одиниць на виході шару порівняння.

Компоненти вектора ваг T_j, пов'язаного з новим запам'ятованим вектором, змінюються таким чином, щоб вони дорівнювали відповідним компонентам вектора C: $t_{ij} = c_i$ (t_{ij} – зв'язок між нейроном $_j$, що виграв, у шарі розпізнавання і нейроном i в шарі порівняння).

Сума може бути розглянута як "розмір" вектора. Якщо вектор «великий», то b_{ij} маленьке. Таким чином, виявляється можливим поділ векторів, якщо один із них є піднабіром іншого (тобто входить до нього).

Приклад. Нехай є вектори X_1 = [1 0 0 0 0], X_2 = [1 1 1 0 0] (тобто X_1 - піднабір X_2).

Якщо у формулі для b_{ij} прибрати суму, то отримаємо

$$T_1 = B_1 = [1\ 0\ 0\ 0\ 0],$$
$$T_2 = B_2 = [1\ 1\ 1\ 0\ 0],$$

Якщо потім подати на вхід мережі вектор X_1, то обидва нейрони будуть збуджені однаково. Якщо ж у формулі для b_{ij} використати суму, то отримаємо

$$B_1 = [1\ 0\ 0\ 0\ 0],$$
$$B_2 = [0{,}5\ 0{,}5\ 0{,}5\ 0\ 0],$$

та при подачі вектора X_1 для нейрона 1 рівень збудження буде відповідати "1", а для нейрона 2 - 0,5 (правильно).

При подачі X_2 отримаємо відповідно 1 та 3/2 (теж правильно).

Нехай далі поданий вектор X_3 = [1 1 0 0 0]. Для нейрона 1 рівень збудження буде одиниця, а для нейрона 2 він становитиме 2/3. Нейрон 1 переможе, вектор набуде значення [1 1 0 0 0]. Величина S = 1/2, і якщо рівень подібності r = 2/3, то нейрон 1 буде загальмований і виграє нейрон 2 (C = [1 1 0 0 0]), S = 1.

Таким чином, мережа АРТ може розпізнавати вхідні вектори, схожі на еталонні образи, коригувати еталонні образи під модифікації вхідних векторів або відстежувати повільні зміни еталона, а також нарощувати число еталонних образів.

Зрозуміло, при моделюванні мереж АРТ на традиційних комп'ютерах важко досягти високої швидкодії, у таких випадках особливо вигідно застосовувати паралельно працюючі обчислювачі.

7 НЕЙРОНІ МЕРЕЖІ КОХОНЕНА

7.1 Структура мережі Кохонена

Нейронні мережі Кохонена [9] призначені на вирішення завдань векторної класифікації. При цьому до навчання не відомо, скільки класів має бути використане для вирішення задачі, тому потрібно використовувати навчання без учителя.

Мережа Кохонена – це ШНС, що містить вхідний шар та шар активних нейронів. Оскільки вхідний шар просто розподіляє вхідні сигнали нейронів активного шару, таку мережу можна вважати одношаровою.

Зазвичай активний шар одномірний або двомірний. Це відображає уявлення нейробіології, відповідно до яких сенсорна інформація, представлена безліччю сигналів, відображається в лінійних або планарних структурах кори головного мозку.

Розрізняють два варіанти мереж Кохонена:

1. *Шар Кохонена*. У ньому нейрони активного шару не впорядковані. У процесі навчання підлаштовуються ваги лише одного нейрона-переможця.

2. *Карта Кохонена* (Self-Organized Map - SOM). У даному разі нейрони активного шару утворюють регулярну структуру. Така карта зазвичай застосовується для кластеризації графічних зображень і звукових сигналів.

У процесі навчання SOM після пред'явлення достатньої кількості образів всі вихідні нейрони мережі розбиваються на підмножини, кожне з яких відгукується на образи відповідного класу.

Важлива властивість SOM після навчання полягає в тому, цим більше схожі об'єкти з навчальної вибірки, тим ближче повинні знаходитися нейрони активного шару, асоційовані з кластерами даних об'єктів. Відстань між нейронами визначається *топологією* мережі. На рис. 7.1 наведено варіанти нейрона з вісьмома або шістьма сусідами.

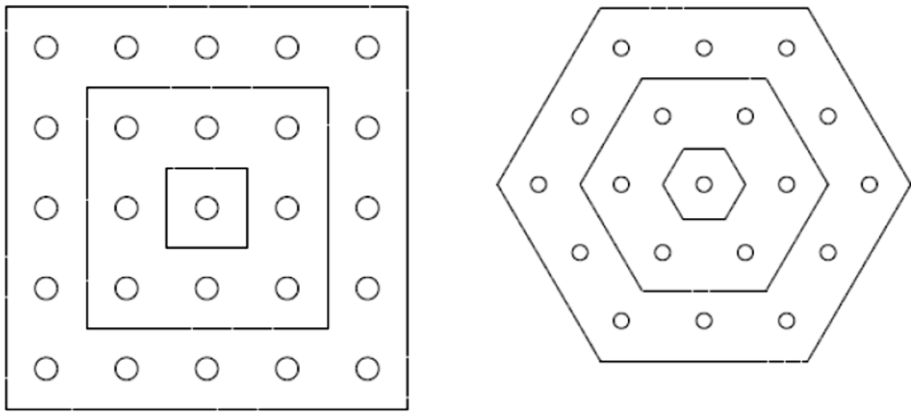


Рисунок 7.1 – Варіанти топології двомірної SOM

Таким чином, нейрони з номерами *i* та *j* характеризуються своїми позиціями *I* та *J*, за допомогою яких можна описати функцію близькості (neighbourhood function), яка може мати вигляд гаусової функції

$$g(i,j) = exp\left(\frac{\|I - J\|^2}{2\sigma^2}\right),$$

де σ - константа, що зменшується в часі.

З допомогою функції близькості описується область взаємодії нейронів шару Кохонена під час навчання. На рисунку 7.2 представлено одновимірну мережу Кохонена з *N* нейронів.

З кожним нейроном Кохонена асоціюється ваговий вектор *W*, якому відповідає точка вхідного простору.

Вхідний шар легко передає компоненти вхідного вектора *X* на кожен нейрон шару Кохонена. Передбачається, що *N* >> *m*.

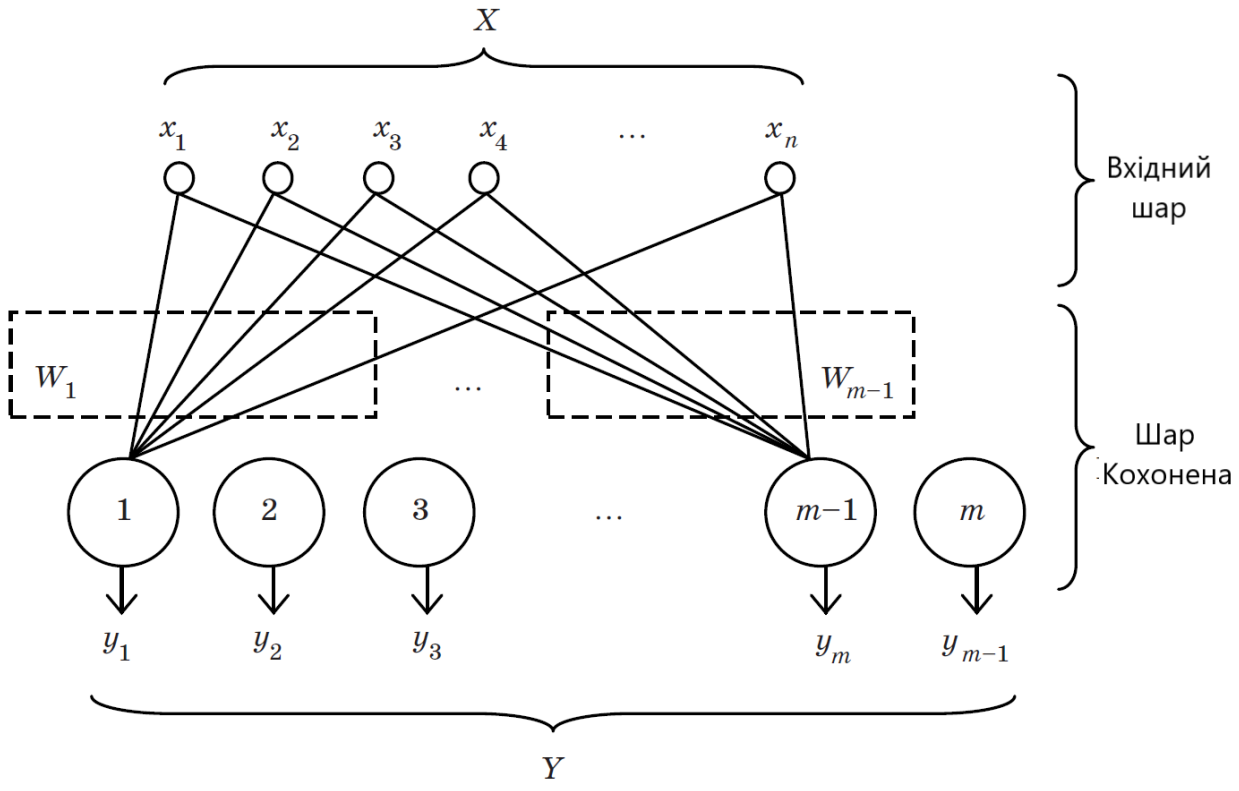


Рисунок 7.2 - Одномірна мережа Кохонена

Кожен *i*-й нейрон 2-го шару має власний вектор ваги W_i, який порівнюється з вхідним вектором *X*. Порівняння передбачає обчислення відстані між *X* і W_i, так що в шарі Кохонена з'являється нейрон-переможець з номером *j*, ваги якого мають найменшу відстань до вхідного вектора:

$$j = argmim_i \|X - W_i\|.$$

В якості метрикb тут може виступати евклідова відстань

$$\|X - W\| = \sqrt{\sum_{i=1}^{n} (x_i - w_i)^2}. \quad (7.1)$$

Якщо вектори *X* і *W* нормалізовані, то мірою близькості можна використовувати скалярний добуток, і тоді вихід нейрона можна описати формулою

$$y_i = W_j X = \sum_{i=1}^{n} w_{ij} x_i, \quad (7.2)$$

та вихід нейрона *j* виявляється максимальним при однакових *X* і *W*:

$$j = argmim_i \|XW_i\|.$$

Нормалізація векторів виконується за формулами

$$w_i^{/} = \frac{w_i}{\sqrt{\sum_{j=1}^{n} (w_j)^2}}, \qquad x_i^{/} = \frac{x_i}{\sqrt{\sum_{j=1}^{n} (x_j)^2}}, \quad i = \overline{1,n}.$$

Таким чином, результатом роботи шару нейронів, що конкурують, при подачі на вхідний шар деякого вектору *X* є визначення нейрона, який має найбільший вихідний сигнал y_j (нейрон-переможець). Цей нейрон має ваговий вектор W_j, який найбільш близький до вхідного вектора.

Нейрони шару Кохонена працюють не ізольовано, між них існують змагальні зв'язки, з допомогою яких близькі нейрони посилюють сигнали одне одного. Формула (7.2) доповнюється другим доданком:

$$y_i = \sum_{i=1}^{n} w_{ij} x_i + \sum_{l \neq j} g(j,l) y_l \quad (7.3)$$

де $g(j,l)$ – сила зв'язку між нейронами, що часто називають *функцією сусідства*.

Можливі різні варіанти $g(j,l)$.

Найпростіший варіант функції сусідства одновимірного шару

$$g(i,j) = \begin{cases} 1, & |i-j| \le r, \\ 0, & |i-j| > r. \end{cases}$$

Гаусова функція сусідства може бути описана у вигляді

$$g(i,j) = exp\left(\frac{R^2}{2\sigma^2}\right),$$

де σ - Вибрана константа; R – відстань між нейронами Кохонена, що обчислюється за формулою

$$R = \left\|r_i - r_j\right\|^2.$$

де r_i та r_j – координати нейронів.

Крім гаусової функції можна використовувати функцію, яка має вигляд «мексиканської капелюхи» (див. рис. 7.3).

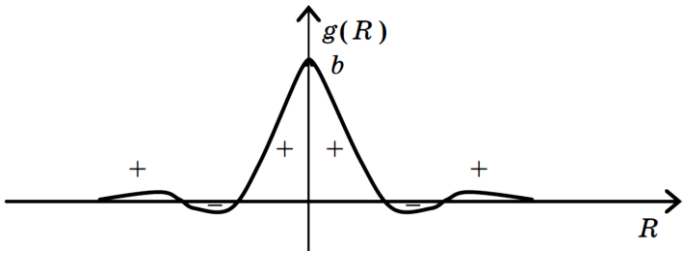


Рисунок 7.3 – Варіант опису взаємодії нейронів

Як випливає з рис. 7.3 навколо точки збудження є невелика околиця, в якій нейрони переходять в активний стан. За межами цієї околиці відбувається гальмування активності нейронів. Спостерігається також слабкий зв'язок активності між нейронами з місця збудження та нейронами з віддалених ділянок.

Насправді можна використовувати спрощений підхід до опису функції взаємодії (див. рис. 7.4).

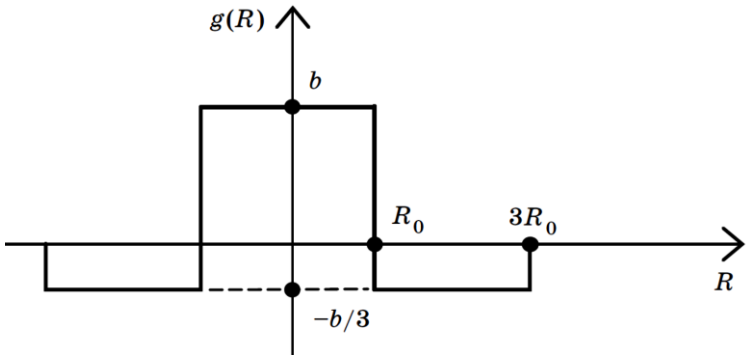


Рисунок 7.4 – Спрощене уявлення сили зв'язків шару Кохонена

Відповідно до рис. 7.4 силу зв'язку можна описати так:

$$g(R) = \begin{cases} b, & R \in [-R_0, R_0], \\ -\frac{b}{3}, & R \in [-R_0, -3R_0], \\ \frac{b}{3}, & R \in [-R_0, 3R_0] \\ 0, & \text{в іншому випадку.} \end{cases},$$

При розгляді мережі з двовимірним шаром Кохонену функції взаємного впливу нейронів мають бути описані для площини (див. рис. 7.5). Можна також розглядати тривимірний шар Кохонена.

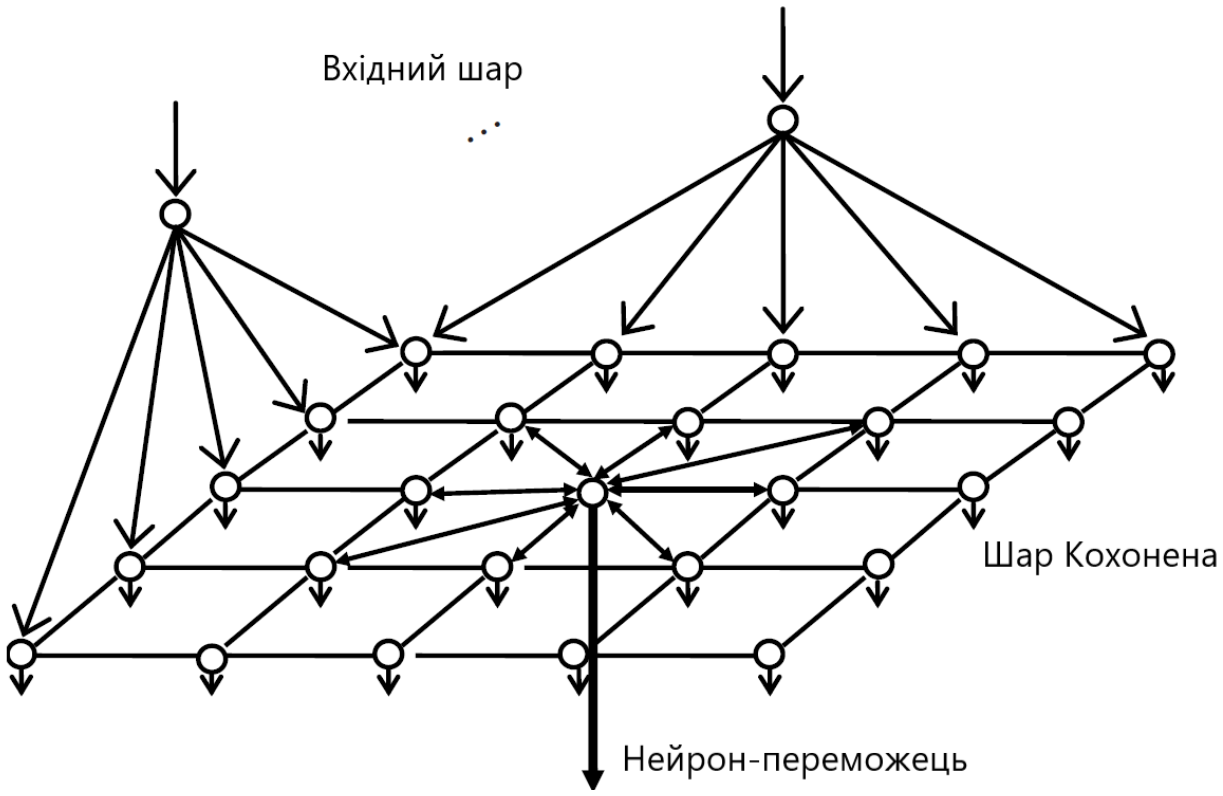


Рисунок 7.5 – ШНС із двовимірним шаром Кохонена

7.2 Навчання мережи Кохонена

Алгоритми навчання мережі Кохонена реалізують навчання без вчителя. Під час навчання *n* вхідних ваг нейрона Кохонена розглядаються як вектор у *n*-мірному просторі. До навчання ваги набувають малі випадкові значення. Потім кожен *n*-вимірний вектор нормалізується у вектор одиничної довжини (з тим самим напрямком, що і вихідний вектор).

При n = 2 всі вхідні вектори можна зобразити на одиничному колі, при n = 3 – на одиничній сфері, а n > 3 слід розглядати поодиноку гіперсферу.

Вхідні вектори навчального набору також нормалізуються, після чого мережа навчається за таким алгоритмом:

1. Подається вхідний вектор X.

2. Визначається нейрон-переможець із номером j за формулою (7.2).

3. Відбувається підстроювання ваг нейронів.

а) якщо навчається шар конкуруючих нейронів, то навчається один нейрон:

$$W_i^T(t+1) = W_i^T(t) + \eta\left(X(t) - W_i^T(t)\right);$$

б) якщо навчається SOM, то відбувається підстроювання ваг нейронів Кохонена за формулою

$$W_i^T(t+1) = W_i^T(t) + \eta g(i,j)\left(X(t) - W_i^T(t)\right);$$

де η - коефіцієнт швидкості навчання.

4. Подається новий вхідний вектор X, і кроки 1–4 повторюються.

Значення η має поступово зменшуватись.

В кінці навчання вплив нейронів один на одного може стати таким малим, щоб навчався всього один ШН, тобто при подачі на вхід мережі деякого вхідного вектору буде збуджуватися лише один нейрон.

Кожна вага, пов'язана з нейроном Кохонена, що виграла, змінюється пропорційною різниці між його величиною і величиною входу, до якого він приєднаний. Для двох входів це можна уявити геометрично (див. рис. 7.6).

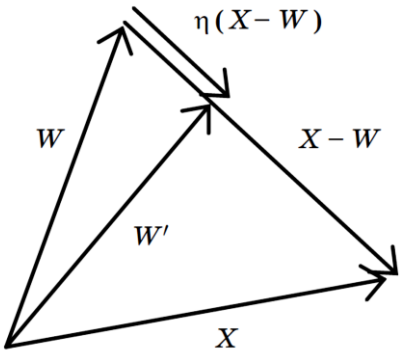


Рисунок 7.6 – Зближення вектору ваг та вхідного вектору

Якби з кожним нейроном Кохонена асоціювався лише один вектор, шар Кохонена міг би бути навчений шляхом простого присвоєння ваг

значень цього вхідного вектору. Але оскільки навчання відбувається за всіма векторами, ваги нейрона отримуються зрештою усередненням за всіма векторами, які повинні його активізувати.

Нормалізовані вхідні та вагові вектори можна розглядати як точки на поверхні одиничної гіперсфери. Вхідні вектори групуються у класи (образи). Якщо переписати ваговий вектор в область, що відповідає центру деякого класу, ШН буде збуджуватися при появі на вході будь-якого вектору цього класу.

Вхідні вектори, яким навчають шар Кохонена, зазвичай розподілені поверхнею гіперсфери не рівномірно, а займають локальні її ділянки. Якщо до навчання вибирати ваги випадково, можна отримати ситуацію, коли більшість ваг буде настільки віддалено від вхідних векторів, що багато нейронів Кохонена завжди матимуть нульовий вихід. Тому необхідно, щоб на околиці кожного вхідного вектору знаходилися вагові вектори. Для того щоб домогтися цього, застосовують *метод випуклої комбінації* [25].

Усі ваги прирівнюються до однієї й тієї величини $1/\sqrt{n}$, де n – розмірність вхідного і вагового векторів, т. е. все компоненти вагових векторів збігаються і вони мають одиничну довжину. Кожній компоненті вектора входу X надається значення

$$\alpha x_i + \left(\frac{1}{\sqrt{n}}\right)(1-\alpha).$$

Спочатку мало α, і всі вхідні компоненти близькі до вагових компонентів. Потім α збільшують, і вхідний вектор стає дедалі більше схожий самого себе. Вагові вектори відстежують один вектор або невелику групу.

Крім методу опуклої комбінації, існують і інші методи. Наприклад, до вхідних векторів можна додавати шуми, щоб кожен нейрон зрештою захопив свій ваговий вектор.

Ще один метод змушує кожен нейрон Кохонена діяти «справедливо», так, щоб нейрони, які збуджуються частіше за інших, були пригнічені, і могли навчитися всі нейрони.

Приклад 7.1. Розглянемо простий приклад навчання шару нейронів, що конкурують. Нехай дані чотири вхідні вектори:

$$X_1 = \begin{bmatrix}1\\1\\0\\0\end{bmatrix}, X_2 = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}, X_3 = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, X_4 = \begin{bmatrix}0\\0\\1\\1\end{bmatrix}$$

Вони мають бути кластеризовані мережею Кохонена, що має два нейрони. Під час навчання змінюються ваги лише нейрона-переможця.

Нехай ваги нейронів отримали деякі випадкові початкові значення (ненормалізовані):

$$W_1^T = \begin{bmatrix} 0{,}2 \\ 0{,}6 \\ 0{,}5 \\ 0{,}9 \end{bmatrix},\ W_2^T = \begin{bmatrix} 0{,}8 \\ 0{,}4 \\ 0{,}7 \\ 0{,}3 \end{bmatrix}.$$

На першому етапі навчання подається вхідний вектор X_1 і обчислюється відстань від нього до вагових векторів (за формулою (7.1)):

$$d_1 = \|X_1 - W_1^T\| = \sqrt{\sum_{i=1}^{4}(x_i - w_i)^2} \approx 1{,}36,$$

$$d_2 = \|X_1 - W_2^T\| = \sqrt{\sum_{i=1}^{4}(x_i - w_i)^2} \approx 0{,}99.$$

Таким чином, переміг виявляється другий нейрон, і його ваги підлаштовуються (η= 0,6):

$$W_2^T = \begin{bmatrix} 0{,}8 \\ 0{,}4 \\ 0{,}7 \\ 0{,}3 \end{bmatrix} + 0{,}6\left[\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0{,}8 \\ 0{,}4 \\ 0{,}7 \\ 0{,}3 \end{bmatrix}\right] = \begin{bmatrix} 0{,}92 \\ 0{,}76 \\ 0{,}28 \\ 0{,}12 \end{bmatrix}$$

На другому етапі навчання подається вхідний вектор X_2:

$$d_1 = \|X_2 - W_1^T\| = \sqrt{\sum_{i=1}^{4}(x_i - w_i)^2} \approx 0{,}81,$$

$$d_2 = \|X_2 - W_2^T\| = \sqrt{\sum_{i=1}^{4}(x_i - w_i)^2} \approx 1{,}5.$$

Переможцем виявляється перший нейрон, і його ваги підлаштовуються:

$$W_1^T = \begin{bmatrix}0{,}2\\0{,}6\\0{,}5\\0{,}9\end{bmatrix} + 0{,}6\left[\begin{bmatrix}0\\0\\0\\1\end{bmatrix} - \begin{bmatrix}0{,}2\\0{,}6\\0{,}5\\0{,}9\end{bmatrix}\right] = \begin{bmatrix}0{,}08\\0{,}24\\0{,}20\\0{,}96\end{bmatrix}$$

Аналогічно подаються вектори X_3 та X_4, коригуються ваги, і перша епоха навчання закінчується.

На другій та наступних епохах значення η поступово зменшується, відбувається наближення ваг до граничних значень:

$$W_1^T = \begin{bmatrix}0\\0\\0{,}5\\1\end{bmatrix}, \qquad W_2^T = \begin{bmatrix}1\\0{,}5\\0\\1\end{bmatrix}$$

Таким чином, центр першого кластера (W_1) виявляється усереднення компонент векторів X_2 і X_4, а центр другого кластера (W_2) - усереднення компонент векторів X_1 і X_3.

7.3 Шар Кохонена

Архітектура змагального шару наведена на рис. 7.7.

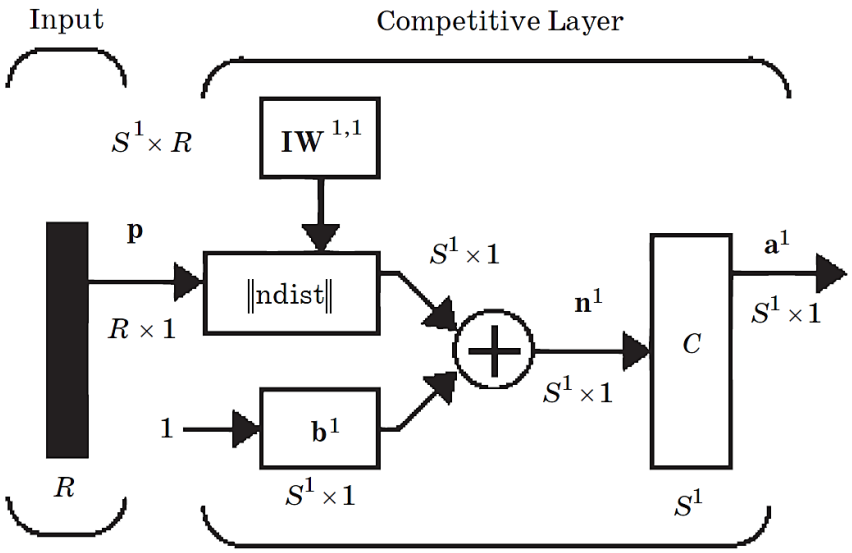


Рисунок 7.7 - Змагальний шар Кохонена у MatLab

На рис. 7.7 R – розмірність вхідного вектору p та векторів ваг; S_1 - число нейронів змагального шару; блок C (competitive layers) означає обчислення виходів нейронів з урахуванням функції близькості. Вектор зсувів b, якій змінюється при навчанні, використовується для охоплення нейронами всього вхідного простору.

Шар Кохонена створюється командою

net = newc(P, S, KLR, CLR),

де *P* – (*R*×*Q*)-матриця мінімальних та максимальних значень *Q* вхідних векторів; *S* – число нейронів; *KLR* та *CLR* – константи швидкості навчання ваг та зміщень.

Проаналізуємо приклади використання шару Кохонена.

Приклад 7.2. Розглянемо створення шару конкуруючих нейронів Кохонена на вирішення попереднього прикладу. Спочатку опишемо вхідні вектори:

```
p=[1 0 1 0; 1 0 0 0; 0 0 0 1; 0 1 0 1]
```

```
p =
     1     0     1     0
     1     0     0     0
     0     0     0     1
     0     1     0     1
```

Потім створимо нейронну мережу:

```
net=newc([ 0 1; 0 1; 0 1; 0 1],2)
```

У цій команді масив вказує діапазони кожної вхідної змінної, а число – число нейронів змагального шару. Початкові значення матриці ваги задаються в центрі вхідних діапазонів.

Задамо кількість епох навчання:

```
net.trainParam.epochs = 500
% Навчання ІНС відбувається за командою
net = train (net, p);
% Перевіримо значення ваг:
ves = net.IW {1,1}
```

```
ves =
    0.0000    0.0000    0.5001    1.0000
    1.0000    0.4996    0.0000    0.0000
```

Значення виявилися близькими до очікуваних для цього прикладу. Перевіримо роботу мережі:

```
p=[0 0 0 1]';
Y = sim (net, p)
```

```
Y =
     0
     1
```

```
p=[1 0 0 0]';
Y = sim (net, p)
```

```
Y =
     1
     0
```

Результат навчання НМ наведені на рисунку 7.8.

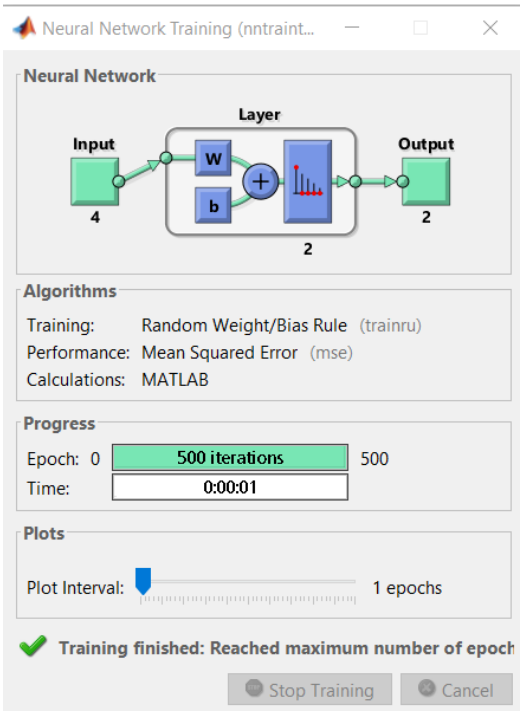


Рисунок 7.8 - Результат навчання НМ Кохонена

Приклад 7.3. Розглянемо завдання кластеризації точок на площині. Розмістимо випадковим чином 60 точок на площині за допомогою команд:

```
A = [rand(1,20) - 0.7; rand(1,20) + 0.2];
B = [rand(1,20) + 0.7; rand(1,20) + 0.7];
C = [rand(1,20) + 0.2; rand(1,20) - 0.7];
plot(A(1,:),A(2,:),'bs')
hold on
plot(B(1,:),B(2,:),'r+')
plot(C(1,:),C(2,:),'go')
grid on
P = [A, B, C];
ncl = 3;
MN = [min(P(1,:)) max(P(1,:)); min(P(:,2)) max(P(:,2))]
net = newc(MN, ncl, 0.1, 0.0005);
net.trainParam.epochs=49;
net.trainParam.show=7;
net = train(net,P);
w = net.IW{1};
plot(w(:,1),w(:,2),'kp');
```

Результат навчання наведено на рис. 7.9 де зірочками позначені центри трьох кластерів.

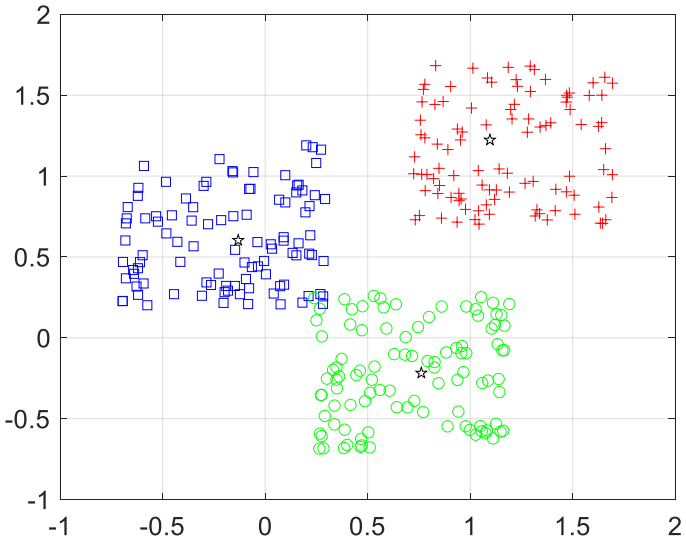


Рисунок 7.9 - Визначення трьох кластерів на площині

Приклад 7.4. У прикладі для створення кластерів використовується спеціальна функція nngenc. З використанням конкурентного навчання

Нейрони в конкурентному шарі вчяться представляти різні області вхідного простору, де виникають вхідні вектори.

P – це набір випадково згенерованих (точки даних) кластеризованих точок тестових даних (див. рис. 7.10). Конкурентна мережа буде використана для класифікації цих точок у класах.

```
% Створюємо входи X.
bounds = [0 1; 0 1];   % Центри кластерів повинні бути в цих межах.
clusters = 8;          % Кількість кластерів.
points = 10;           % Кількість точок у кожному кластері.
std_dev = 0.05;        % Стандартне відхилення кожного кластера.
x = nngenc(bounds,clusters,points,std_dev);
% Вхідні дані графіка X.
plot(x(1,:),x(2,:),'+r');
title('Input Vectors');
xlabel('x(1)');
ylabel('x(2)');
```

Результат навчання наведено на рис. 7.9 де зірочками позначені центри трьох кластерів

Тут competlayer приймає два аргументи: кількість нейронів та швидкість навчання.

Можливо налаштувати мережеві входи (зазвичай це робиться автоматично у train) і побудувати початкові вектори ваги, щоб побачити їхню спробу класифікації.

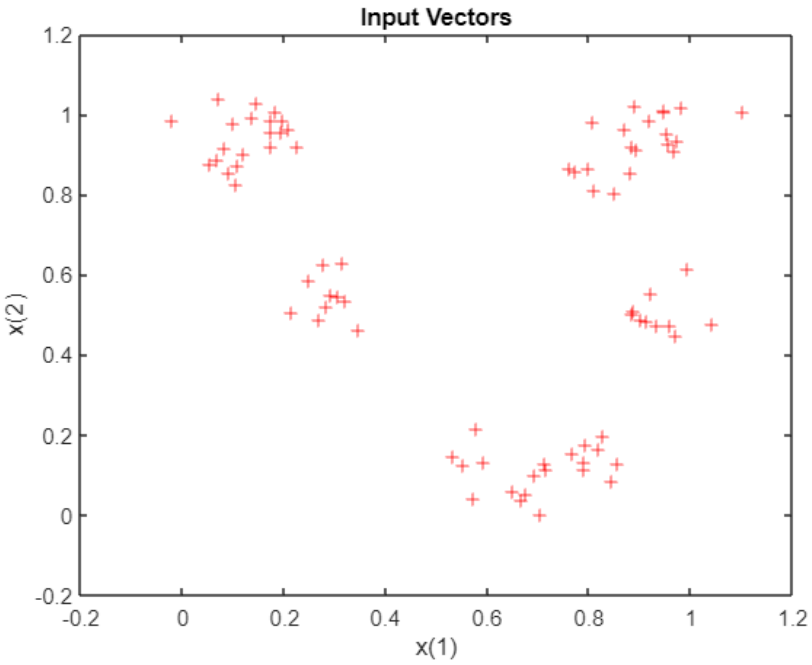


Рисунок 7.10 - Набір випадково згенерованих (точки даних) кластеризованих точок тестових даних

Вагові вектори (o) будуть навчені так, щоб вони розташовувалися в центрі кластерів вхідних векторів. (+'s) (див. рис. 7.11)

```
net = competlayer(8,.1);
net = configure(net,x);
w = net.IW{1};
plot(x(1,:),x(2,:),'+r');
hold on;
circles = plot(w(:,1),w(:,2),'ob');
```

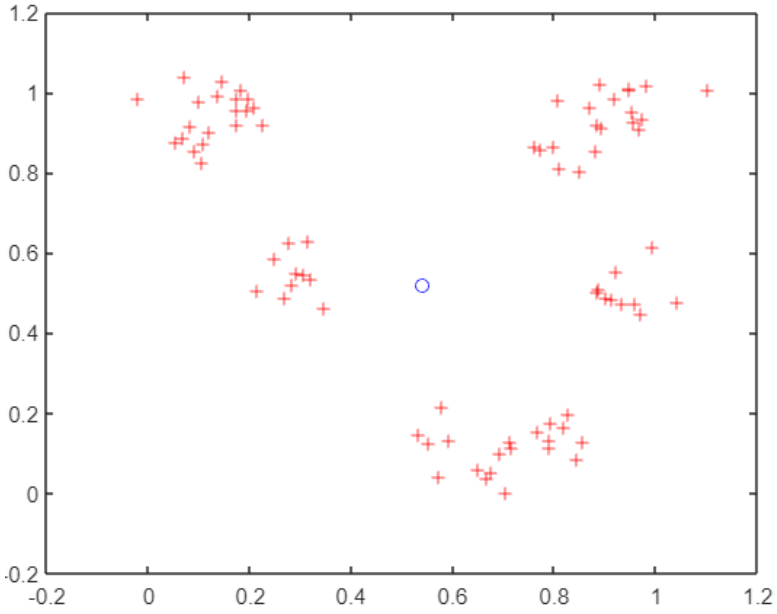


Рисунок 7.11 - Центрі кластерів вхідних векторів

Встановлюємо кількість епох для тренування перед зупинкою та тренуємо конкурентний рівень (може зайняти кілька секунд).

Наносимо оновлені ваги шарів на той же графік. Результат наведено на рис. 7.12.

```
et.trainParam.epochs = 7;
net = train(net,x);
w = net.IW{1};
delete(circles);
plot(w(:,1),w(:,2),'ob');
```

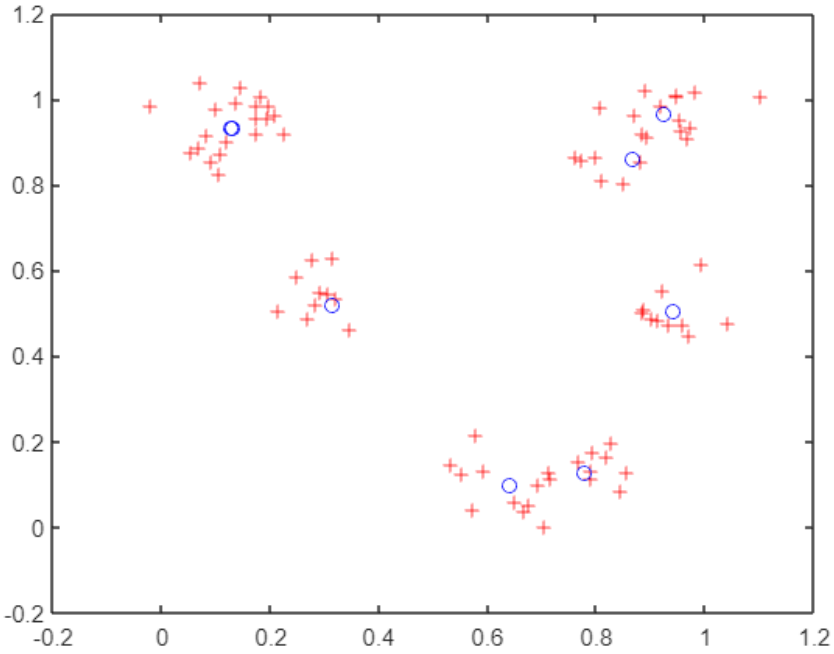


Рисунок 7.12 - Визначення 7 кластерів на площині

Тепер ми можемо використовувати конкурентний рівень як класифікатор, де кожен нейрон відповідає іншій категорії. Визначаємо вхідний вектор X_1 як [0; 0,2].

Вихід Y вказує на те, який нейрон відповідає, тобто, до якого класу він належить вхід.

```
x1 = [0; 0.2];
y = net(x1)
      y = 8×1
          0
          1
          0
          0
          0
          0
          0
          0
```

Приклад 7.5. У аналізованому прикладі кластеризація виконується у просторі.

```
d1 = randn(3,20); % кластер із центром на початку координат
d2 = randn(3,20) + 3; % кластер з центром (3, 3, 3)
d3 = randn(3,20); d3(1,:) = d3(1,:) + 9; % кластер р центром (9, 0, 0)
d = [d1 d2 d3]; % масив для кластеризації
plot3(d(1,:), d(2,:), d(3,:), 'ko'), hold on, box on
net = newc(minmax(d), 3);
net.trainParam.epochs = 100;
net = train(net, d);
gcf,
w = net.IW{1};
plot3(w(:,1),w(:,2),w(:,3),'rp');
```

Результат кластеризації наведено на рис. 7.13 (центрам кластерів відповідають зірочки).

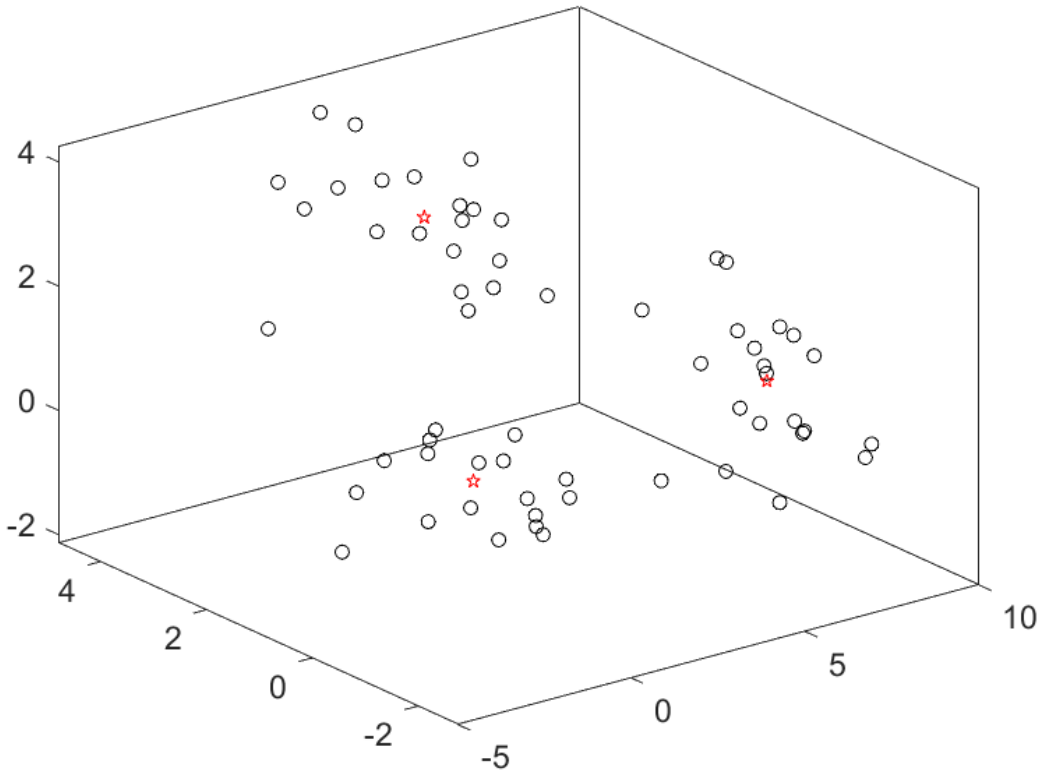


Рисунок 7.13 - Визначення трьох кластерів у просторі

7.4 Самоорганізовані карти Кохонена

Картки Кохонена, що самоорганізуються, застосовуються для візуалізації багатовимірних даних. Звісно, спроектувати багатовимірну вибірку на площину без спотворень неможливо, і SOM відбивають лише загальну картину. Однак використання SOM дозволяє аналізувати

особливості кластерної структури багатовимірної вибірки. Це дуже важливо для якісного аналізу інформації.

Ідея полягає в тому, щоб спроектувати всі об'єкти вибірки на плоску карту, точніше на безліч вузлів прямокутної сітки заздалегідь заданого розміру *M* × *N*, які можуть мати порядок десятків або сотень. Для того щоб карта відображала кластерну структуру вибірки, близькі об'єкти повинні потрапляти в близькі вузли сітки. Нейрони SOM упорядковуватимуться так, щоб мати рівномірний розподіл при рівномірному розподілі вхідних векторів. Якщо ж вхідні вектори розподілені нерівномірно, то й нейрони прагнуть потрапити до центру кластерів.

Таким чином, SOM вирішує два завдання:

- зниження розмірності даних із мінімальною втратою інформації (аналіз основних компонент даних, виділення наборів незалежних ознак);

- зменшення різноманітності даних шляхом виділення кінцевого набору прототипів та віднесення даних до одного з таких типів (тобто кластеризація).

Самоорганізована карта Кохонена створюється за допомогою команди

net = newsom(PR,[d1,d2,...], tfcn, dfcn, olr, osteps, tlr, tnd),

де PR – (R×2)-матриця мінімальних та максимальних значень R вхідних елементів; di - розмірність шару (за умовчанням [5 8]); tfcn – вибір топології (за умовчанням 'hextop'); dfcn – функція дистанції (за умовчанням 'linkdist'); olr – параметр швидкості навчання на етапі розміщення (за умовчанням 0,9); osteps – число циклів навчання на етапі підстроювання (за умовчанням 1000); *tlr* – параметр швидкості на етапі підстроювання (за умовчанням 0,02); *tnd* – розмір околиці на етапі підстроювання (за умовчанням 1).

Функція hextop відповідає гексагональній, gridtop – прямокутній та randtop – випадковій топології.

Функція дистанції dist відповідає евклідовій відстані, boxdist – максимальне координатне усунення, mandist – відстань сумарного координатного усунення, linkdist – відстань зв'язку.

Налаштування SOM здійснюється за кожним вхідним вектором. Насамперед визначається нейрон-переможець та коригуються його вектор ваг і вектори сусідніх нейронів згідно зі співвідношенням

$$dW = lr \cdot A2 \cdot (P^* - W),$$

де *lr* - параметр швидкості навчання, що дорівнює *olr* для етапу впорядкування нейронів і *tlr* - для етапу підстроювання; *A*2 – масив

сусідства для нейронів, розташованих на околиці нейрона-переможця з номером *i*:

$$A2(i,q) = \begin{cases} 1, & a(i,q) = 1, \\ 0{,}5, & a(i,q) = 1, D(i,j) \le nd, \\ 0, & \text{в інших випадкахю} \end{cases}$$

Тут *a(i,q)* – елемент виходу нейронної мережі; *D(i, j)* – відстань між нейронами *i* та *j*; *nd* – розмір околиці нейрона-переможця.

Таким чином, вага нейрона-переможця змінюється пропорційна половинному параметру швидкості навчання, а ваги сусідніх нейронів - пропорційною половинному значенню цього параметру.

Весь процес навчання карти Кохонена поділяється на етапи:

- упорядкування векторів вагових коефіцієнтів у просторі ознак;
- підстроювання ваг нейронів по відношенню до набору векторів входу.

На етапі впорядкування використовується фіксована кількість кроків. Початковий розмір околиці призначається рівним максимальній відстані між нейронами для обраної топології і потім зменшується до величини, що використовується на наступному етапі. Він обчислюється за формулою

$$nd = 1{,}00001 + (\max(d) - 1)\left(1 - \frac{s}{S}\right),$$

де *max*(*d*) – максимальна відстань між нейронами; *s* – номер поточного кроку; *S* – число циклів на етапі упорядкування. Параметр швидкості навчання змінюється за правилом

$$lr = trl + (olr - tlr)\left(1 - \frac{s}{S}\right).$$

На етапі підстроювання, який триває протягом частини процедури навчання, розмір околиці залишається постійним і рівним

$$nd = tnd + 0{,}00001,$$

а параметр швидкості навчання змінюється за таким правилом:

$$lr = trlS/s.$$

Параметр швидкості навчання продовжує зменшуватись, але дуже повільно. Мале значення околиці та повільне зменшення параметра швидкості навчання добре налаштовують мережу за збереження розміщення, знайденого попередньому етапі. Кількість кроків на етапі

підстроювання має значно перевищувати кількість кроків на етапі розміщення. На цьому етапі відбувається тонка настройка ваг нейронів по відношенню до набору векторів входів.

Нейрони карти Кохонена впорядковуються так, щоб при рівномірній щільності векторів входу дані нейрони також були рівномірно розподілені. Якщо вектори входу розподілені нерівномірно, і нейрони набувають тенденцію розподілятися відповідно до щільністю розміщення векторів входу.

Приклад 7.6. Апроксимація параболи одновимірним шаром з десяти нейронів:

```
x=-1:0.05:1;
y=x.*x;
P = [x; y];
net=newsom([0 1;0 1],[10]);
net.trainParam.epochs =1000;
net1=train(net,P);
plotsom(net1.iw{1,1},net1.layers{1}.distances)
```

Результат навчання SOM наведено на рис. 7.14.

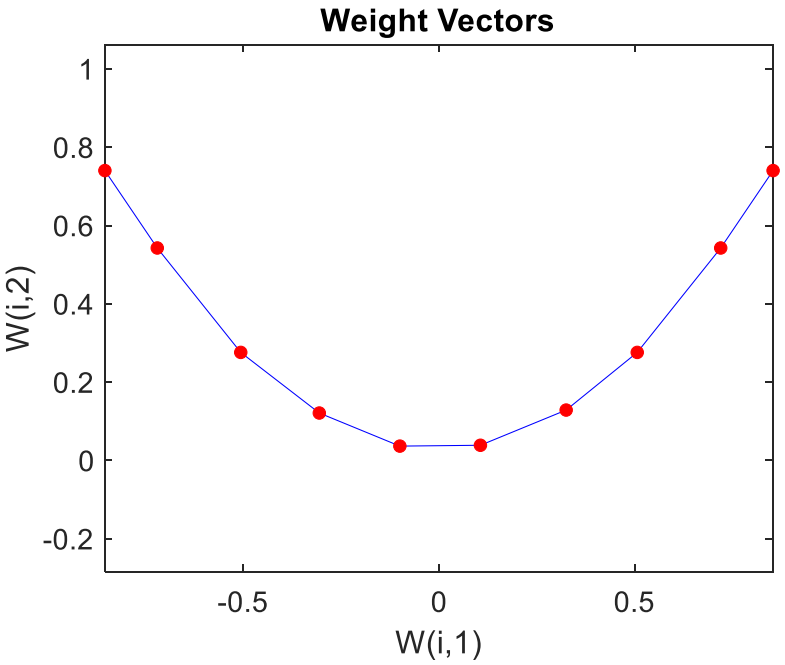


Рисунок 7.14 - Використання одновимірної SOM

При подачі на вхід мережі конкретного вектору можна дізнатися номер нейрона, що спрацював.

```
a=sim(net,[-0.6;0.4])
    a =
        1
        0
        0
```

0
0
0
0
0
0
0

Приклад 7.7. Кластеризація випадково заданого масиву з 40 точок на площині

```
P = [rand(1,40)*2; rand(1,40)];
net = newsom([0 2; 0 1],[3 5]);
net = train(net,P);
plot(P(1,:),P(2,:),'.g','markersize',20)
hold on
plotsom(net.iw{1,1},net.layers{1}.distances)
```

Результат кластеризації наведено на рис. 7.15. MatLab має додаткові засоби аналізу SOM. Функція

```
plotsompos(net,P);
```

породжує графік, аналогічний показаному на рис. 7.15. Функція

```
plotsomnd(net);
```

дозволяє візуалізувати дистанцію між вагами нейронів. Приклад наведено на рис. 7.16.

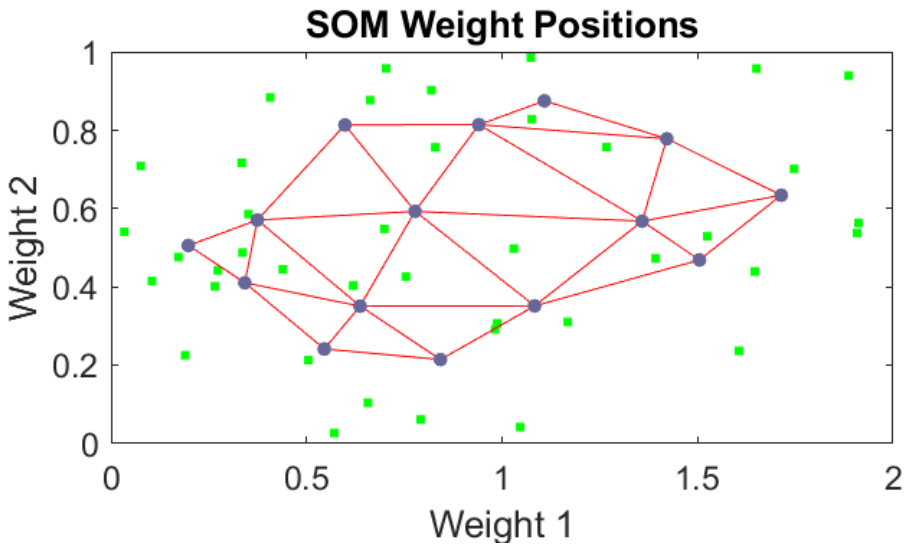


Рисунок 7.15 – Використання двовимірної SOM

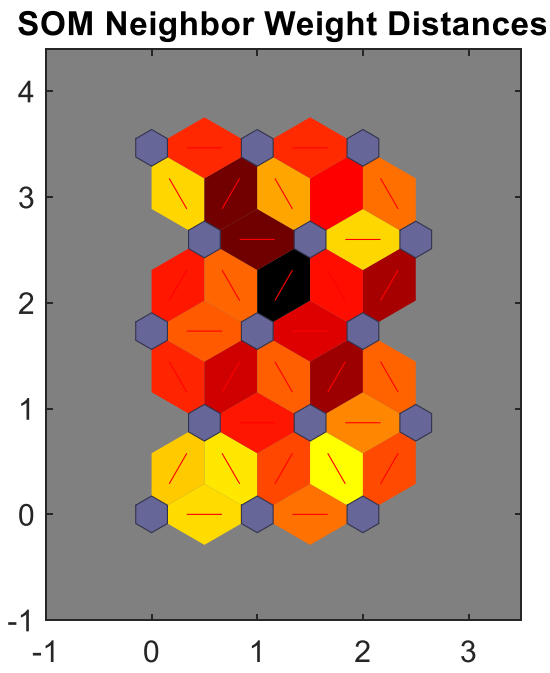


Рисунок 7.16 - Візуалізація дистанції між вагами SOM

Ще одна функція дозволяє дізнатися кількість вхідних векторів, на які реагує кожен нейрон:
>> plotsomhits(net2,P);

Приклад наведено на рис. 7.17

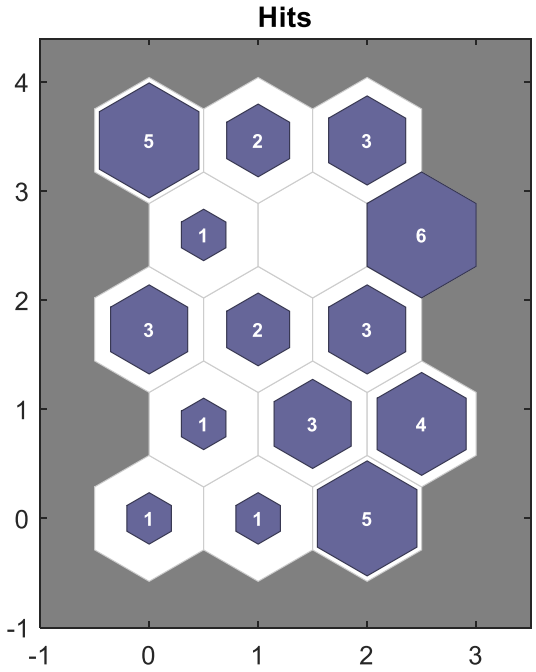


Рисунок 7.17 – Візуалізація числа векторів, асоційованих з нейронами двовимірною SOM

Зручним інструментом візуалізації даних є розмальовка топографічних карт аналогічно до того, як це роблять на звичайних географічних картах. Кожна ознака даних породжує своє забарвлення осередків карти – за середнім значенням цієї ознаки у даних, які у цей осередок.

7.5. Нейронні мережі класифікації

Нейронні мережі для класифікації вхідних векторів або LVQ-мережі (Learning Vector Quantization) виконують і кластеризацію, і класифікацію вхідних векторів. Це виявляється можливим завдяки додаванню до шару Кохонен ще одного шару - лінійного (рис. 7.18).

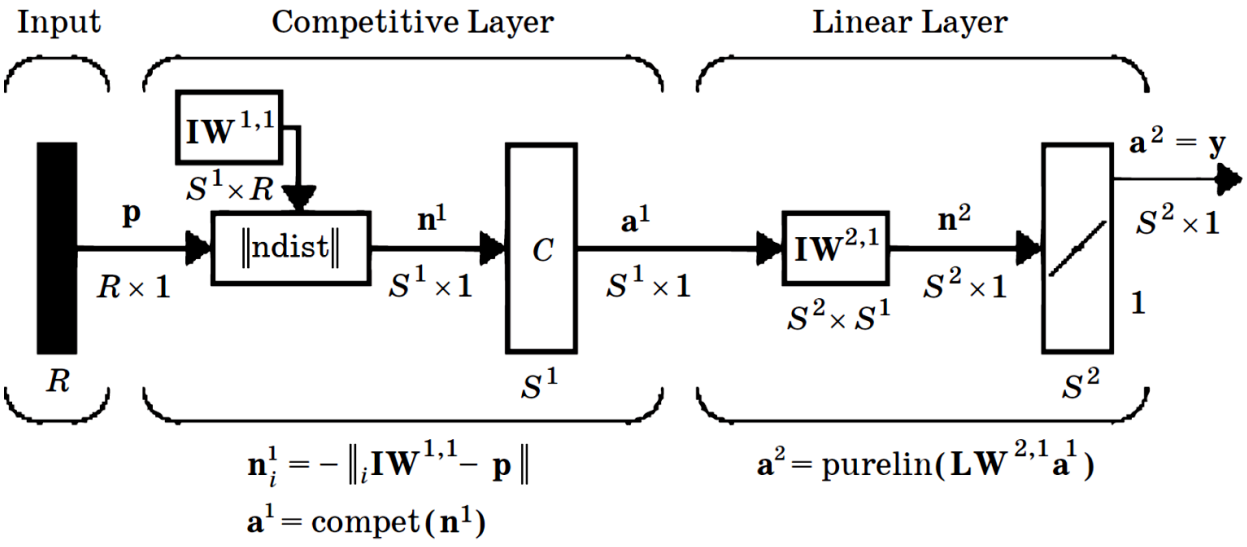


Рисунок 7.18 - Структура LVQ-мережі

Шар Кохонена (конкуруючий шар) здатний підтримувати до S^1 кластерів (тобто один нейрон однією кластер).

Лінійний шар відповідає класам S^2 (де $S^2 \le S^1$). При цьому до одного класу можуть належати кілька кластерів.

Створення LVQ-мережі відбувається за допомогою команди

net = newlvq(PR, S1, PC, LR, LF),

де *PR* – масив *R*×2 мінімальних та максимальних значень для R елементів вектора входу; S^1 – число нейронів конкуруючого шару; *PC* – вектор із S^2 елементами, що визначають відсоткову частку приналежності вхідних векторів до відомого класу; *LR* – параметр швидкості налаштування; *LF* – ім'я функції налаштування.

Оскільки наперед відомо, як кластери 1-го шару співвідносяться з цільовими класами 2-го шару, це дозволяє заздалегідь задати елементи ваги матриці останнього. Однак, щоб знайти правильний

кластер для кожного вектора навчальної множини, необхідно виконати процедуру навчання мережі.

Навчання LVQ-мережі відбувається з урахуванням навчальної послідовності, у якій кожен цільової вектор має єдиний елемент, рівний одиниці, інші елементи рівні нулю:

$$\{p_1, t_1\}, \{p_2, t_2\}, \cdots, \{p_N, t_N\}.$$

Приклад 7.8. Дано десять векторів (точок на площині). Потрібно розділити їх на два класи відповідно до цільового вектора *T*:

```
P = [-3 -2 -2 0 0 0 0 2 2 3; 0 1 -1 2 1 -1 -2 1 -1 0];
Tc = [1 1 1 2 2 2 2 1 1 1]; % Індекси класів;

I1 = find(Tc==1); % вектор індексів першого класу
I2 = find(Tc==2); % вектор індексів другого класу

figure(1), clf, axis([-4,4,-3,3]), hold on
plot(P(1,I1),P(2,I1),'+r')
plot(P(1,I2),P(2,I2),'xb')

T = ind2vec(Tc); % розряджена цільова матриця;
T = full(T); % повна цільова матриця;

net = newlvq(minmax(P), 4, [0.6 0.4]);
net.trainParam.epochs = 200;
net.trainParam.lr = 0.05;
net.trainParam.goal = 1e-5;
net = train(net,P,T);

w = net.IW{1};
plot(w(:,1),w(:,2),'rp');
```

Результат навчання наведено на рис. 7.19 де зірочки відповідають центрам кластерів. Перевірити роботу мережі можна за допомогою команд

```
Y = sim(net,P);
Yc = vec2ind(Y)

Yc =
    1    1    1    2    2    2    2    1    1    1
```

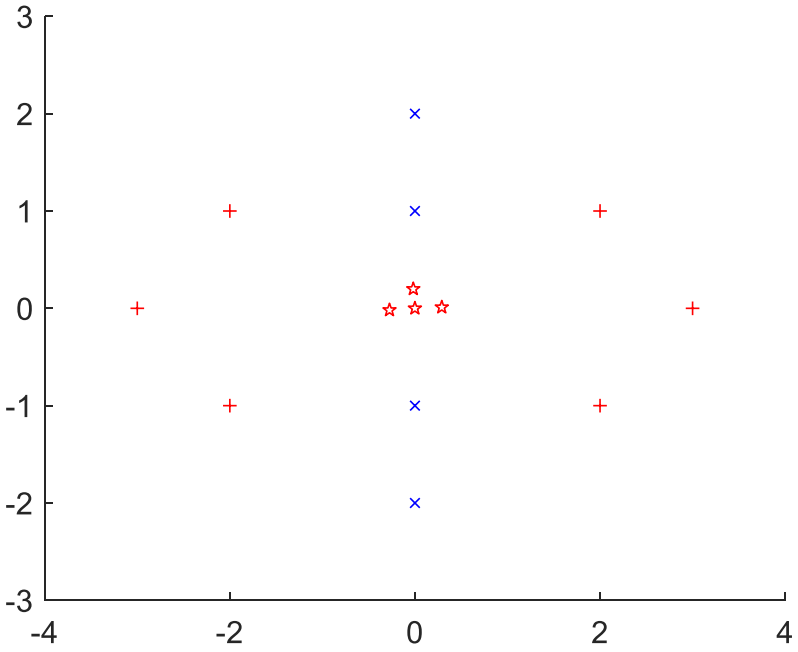


Рисунок 7.19 – Результат навчання LQV-мережі

Приклад 7.9. Нехай на площині задано кілька груп точок, які потрібно розбити на два класи, позначені хрестиками та кружками (див. рис. 7.20):

```
A = [rand(1,20) + 0.2; rand(1,20) + 0.2];
B = [rand(1,20) + 1.2; rand(1,20) + 1.2];
C = [rand(1,20) + 2.2; rand(1,20) + 2.2];
D = [rand(1,20) + 3.2; rand(1,20) + 3.2];
plot(A(1,:),A(2,:),'go')
hold on
plot(B(1,:),B(2,:),'r+')
plot(C(1,:),C(2,:),'go')
plot(D(1,:),D(2,:),'r+')
grid on
% Сформуємо дані для навчання:
t1=ones([1 20]); t2=t1+t1;
Tc=[t1 t2 t1 t2];
T = ind2vec(Tc);
P = [A B C D];
% Визначимо LVQ-мережу та запустимо процес навчання:
net = newlvq(minmax(P), 4, [0.5 0.5]);
net.trainParam.epochs = 500;
net = train(net,P,T);
% Перевірку можна виконати командами (див. рис. 7.21)
Y = sim(net,P);
Yc = vec2ind(Y);
```

```
figure; hold;
 for i=1:length(P)
 plot(P(1,i), P(2,i), ['.' colors(Yc(i))]);
 end
plotsom(net.iw{1,1}')
```

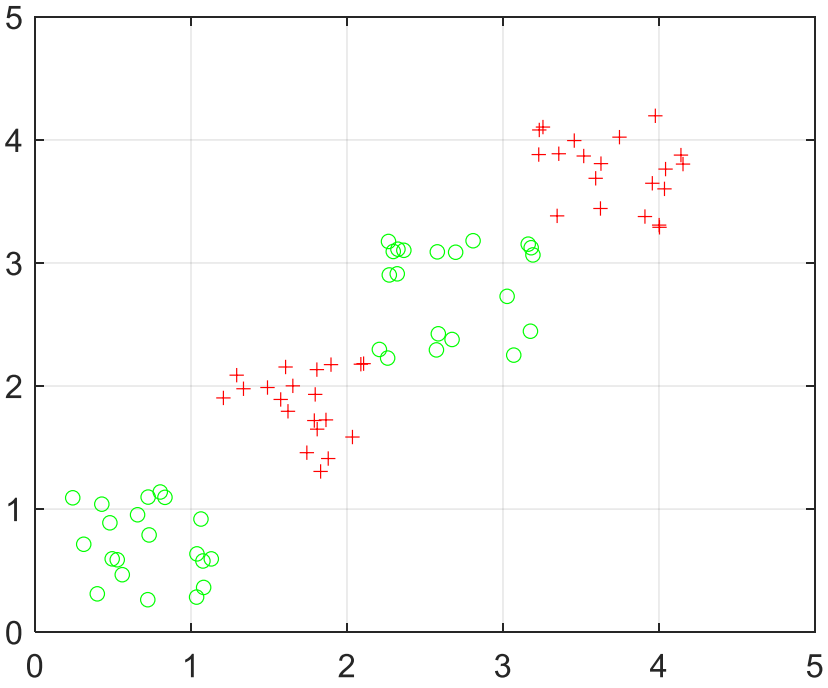


Рисунок 7.20 - Вихідні дані для класифікації

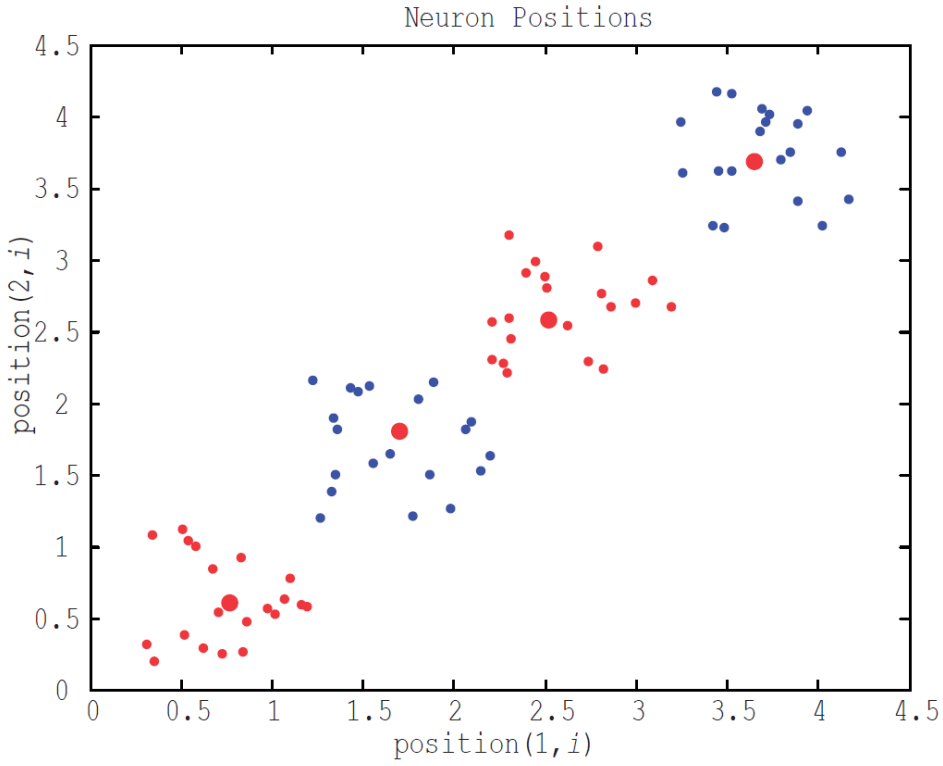


Рисунок 7.21 - LVQ-мережа в режимі класифікації

8 НЕЙРОКЕРУВАННЯ

8.1. Управління зі зворотним зв'язком

Розглянемо традиційну систему управління з негативним зворотним зв'язком (рис.8.1).

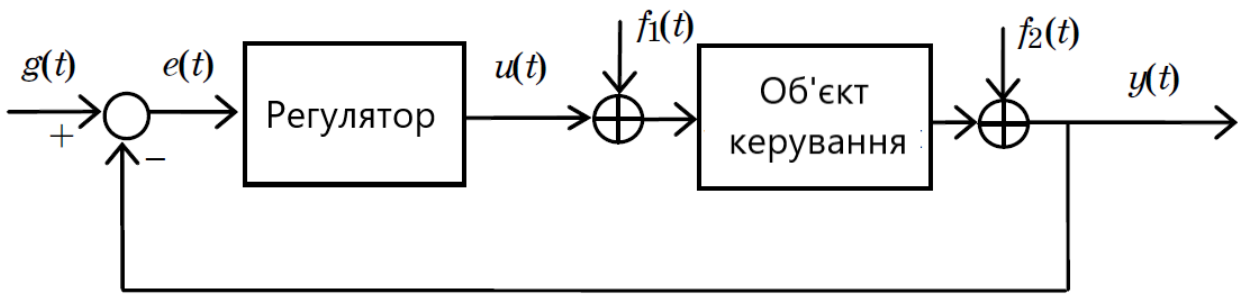


Рисунок 8.1 - Система управління із зворотним зв'язком

Вплив $g(t)$ визначає бажане значення виходу об'єкта управління $y(t)$. Помилка керування $e(t) = g(t) – y(t)$ використовується регулятором для формування сигналу керування $u(t)$. Об'єкт являє собою динамічну систему, яка може бути нестійкою або мати погані динамічні властивості. Регулятор служить поліпшення динамічного поведінки об'єкта. Шуми на вході та виході об'єкта $f_1(t)$ та $f_2(t)$ можуть бути одним із джерел неточності (нечіткості) завдання управління (при синтезі регулятора ними часто нехтують).

Класична теорія управління орієнтована переважно на синтез лінійних регуляторів для таких об'єктів. Властивість лінійності дозволяє отримати зручні алгоритми оцінки таких важливих системних властивостей, як стійкість, керованість, спостережуваність тощо.

Проте всі реальні об'єкти є нелінійними, і лінеаризація їхнього математичного опису вимагає розгляду певної робочої точки, щодо якої розглядаються малі відхилення.

Нелінійність математичної моделі виявляється у присутності нелінійних блоків: насичення, сухе тертя, гістерезис тощо. Зокрема, введення у структуру рис. 11.1 обмеження значення сигналу управління робить її нелінійною. Нелінійність також виникає, якщо в математичному описі є нелінійні функції (квадрат, квадратний корінь, тригонометричні функції тощо).

Нечіткі логічні регулятори (НЛР), нелінійні за своєю суттю, можуть керувати лінійними об'єктами краще, ніж класичні регулятори, і навіть керувати нелінійними об'єктами, котрим лінійні регулятори непридатні.

Для лінійних скалярних об'єктів (з одним входом та одним виходом) при описі перехідних процесів традиційно розглядається

реакція $y(t)$ на стрибкоподібний вхідний вплив $g(t)$, і використовуються такі параметри, як (рис. 8.2):

− час наростання $t_н$, тобто час, за який змінна $y(t)$ зростає з 0.1 до 0.9 значення $y_{вст}(t)$, що встановилося;

− перерегулювання $d = (y_{max}(t) – y_{вст}(t))/100$ %;

− помилка, що встановилася $e = g(t) − y_{вст}(t)$;

− час перехідного процесу $t_{пп}$ (час від початку перехідного процесу до моменту, коли $y(t)$ не залишає інтервал 1 ± 0.05;

− $t_з$ – час загасання перехідного процесу (час між моментом першого досягнення сигналом $y(t)$ одиничного рівня та моментом, починаючи з якого значення y(t) залишаються всередині інтервалу [1±0.05].

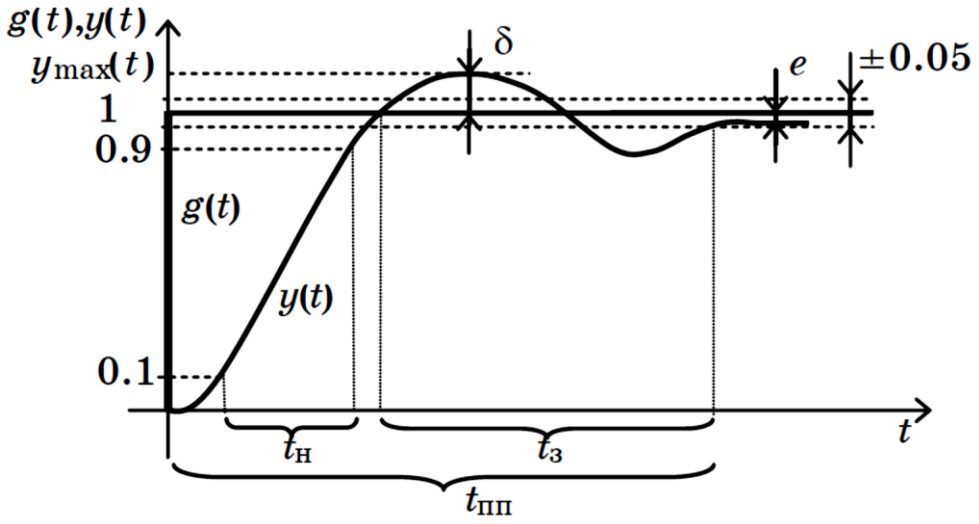


Рисунок 8.2 – Параметри перехідного процесу

Реакція об'єкта на ступінчасту вхідну дію називається *перехідною функцією або кривою розгону*.

За характером значення вихідної величини об'єкта (при подачі на вхід одиничного стрибка) всі об'єкти діляться на дві групи: з самовирівнюванням і без самовирівнювання.

Самовирівнюванням називається властивість регульованого об'єкта після зміни вхідного сигналу самостійно повернутися до нового стану (див. рис. 11.2). Самовирівнювання полегшує роботу регулятора. Стійке функціонування об'єкта управління без самовирівнювання неможливе без регулятора.

8.2 Моделі об'єктів керування

Як показує багаторічний досвід конструювання промислових систем управління, більшість реальних динамічних об'єктів або описується передаточними функціями першого або другого порядку

(можливо – із запізненням), або їх динамічні характеристики можуть бути апроксимовані цими функціями. Таким чином, типовий опис об'єкта управління має вигляд:

$$W(p) = \frac{ke^{-\tau p}}{(T_1 p + 1)(T_2 p + 1)};\ T_1 > 2T_2. \qquad (8.1)$$

За наявності ланки запізнення ($e^{(-\tau p)}$) порядок передаточної функції зростає на k разів, де k – порядок розкладання експоненційної функції ряд Паде.

Для об'єктів керування з явно вираженою постійною часу передатна функція спрощується:

$$W(p) = \frac{ke^{-\tau p}}{Tp + 1}, \qquad (8.2)$$

де k, T, τ– коефіцієнт посилення, постійна часу та запізнення, які мають бути визначені в околиці номінального режиму роботи об'єкта.

Для об'єкта керування без самовирівнювання передаточна функція має вигляд

$$W(p) = \frac{ke^{-\tau p}}{p}. \qquad (8.3)$$

Виникає завдання визначення параметрів динамічної моделі об'єкта керування (ідентифікації). Це можна зробити експериментально на реальному об'єкті управління. Можуть розглядатися часові або частотні характеристики об'єкта управління.

Розглянемо експериментальні методи визначення часових динамічних характеристик об'єкта управління. Вони поділяються на два класи: активні та пасивні.

Активні методи передбачають подачу на вхід об'єкта пробних сигналів, таких як ступінчастий або прямокутний імпульс.

Залежно від виду пробного сигналу вибирають відповідні способи обробки вихідного сигналу об'єкта управління.

Так, наприклад, при подачі ступінчастого керуючого сигналу знімають криву розгону об'єкта, а при подачі прямокутного імпульсного сигналу знімають криву відгуку. Крива відгуку знімається для об'єктів, які не допускають подачі на вхід об'єкта ступеневих сигналів.

Пасивні методи застосовуються тоді, коли неможливе порушення нормального перебігу технологічного процесу.

Розглянемо визначення динамічних характеристик об'єкта виду 8.2) щодо його кривої розгону при подачі ступінчастого пробного сигналу.

У початковий момент необхідно, щоб система управління перебувала в спокої, тобто регульована величина $y(t)$ і вплив $g(t)$, що задає вплив, не змінювалися, а зовнішні обурення були відсутні. Потім на вхід виконавчого механізму подається ступінчаста дія, і стан об'єкта починає змінюватися.

При знятті кривої розгону необхідно виконати низку умов:

1) якщо проектується система стабілізації, то крива розгону має зніматися на околиці робочої точки процесу;

2) криві розгону необхідно знімати як за позитивних, і негативних стрибках управляючого сигналу. По виду кривих можна будувати висновки про ступеня асиметрії об'єкта;

3) за наявності зашумленого виходу бажано знімати кілька кривих розгону з їх подальшим накладенням один на одного та отриманням усередненої кривої.

Знявши криву розгону і оцінивши характер об'єкта управління (з самовирівнюванням або без) можна визначити параметри відповідної передаточної функції. Зокрема, з цією метою можна використовувати метод дотику до точки перегину кривої розгону.

Розглянемо приклад використання методу для об'єкта з самовирівнюванням (рис. 8.3).

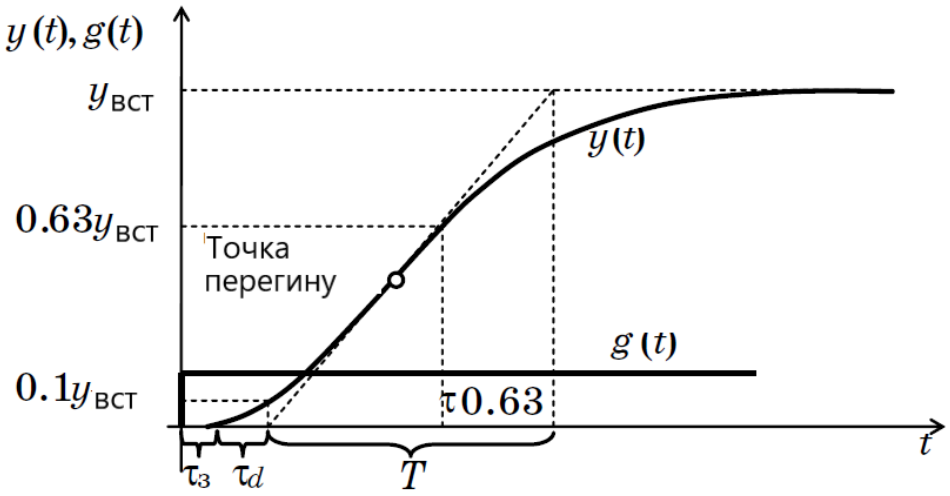


Рисунок 8.3 – Графік кривої розгону

Точка перегину відповідає переходу кривої від режиму прискорення до режиму уповільнення темпу наростання вихідного сигналу. Постійна часу T визначається відповідно до графіка рис. 8.3.

Досить точною вважається також оцінка виду

$$T = \tau_{0.63} - \tau_з$$

Динамічне запізнення τ також визначається за графіком, і складається з двох компонентів:

$$\tau = \tau_з + \tau_d,$$

де $\tau_з$ - чисте (ємнісне) запізнення, τ_d – транспортне запізнення.

Для об'єкта з самовирівнюванням динамічний коефіцієнт посилення *K* показує, у скільки разів ця ланка посилює вхідний сигнал (див. рис. 8.3):

$$K = \frac{y_{\text{вст}}}{g}.$$

Для об'єкта без самовирівнювання виду (8.3) динамічний коефіцієнт посилення *K* визначається як відношення швидкості зміни вихідної величини у до величини стрибка вхідного сигналу. При одиничному стрибку

$$K = \frac{\Delta y}{\Delta t}.$$

Для визначення динамічних показників об'єкта виду (8.1) можна використовувати метод Орманса.

Цей метод дозволяє за нормованою кривою розгону визначити дві домінуючі постійні об'єкти управління.

Перед початком обробки перехідної характеристики потрібно її пронормувати (діапазон зміни нормованої кривої 0-1) і виділити з початкової ділянки величину чистого тимчасового запізнення.

Нормована крива перехідна характеристика визначається формулою

$$h(t) = \frac{y(t) - y_{min}}{y_{max} - y_{min}}.$$

Потім визначається час, що відповідає значенню y_H = 0.7 і позначається t_7. Отриманий інтервал поділяється на три частини. Піднімається перпендикуляр до кривої розгону та визначається величина y_{H4}. Аналітично доведено зв'язок між точками кривої розгону та параметрами моделі, а саме

$$t_7 = 1.2(T_1 + T_2);\ \ t_4 = \frac{t_7}{3}.$$

Постійні часу об'єкта управління T_1 та T_2 визначаються за допомогою допоміжної величини Z^2, для якої використовується номограма (рис. 8.4).

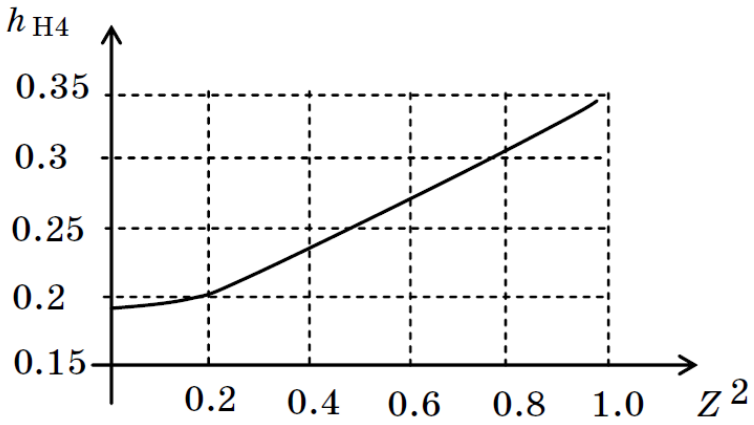


Рисунок 8.4 – Номограма для визначення величини

Постійні часу об'єкта управління T_1 і T_2 визначаються за такими формулами:

$$T_1 = \frac{t_7}{2.4}(1+z);\ T_2 = \frac{t_7}{2.4}(1-z)$$

Якщо $h_{H4} < 0.19$, то визначення динаміки об'єкта використовують метод площини. Якщо $T_1 >> T_2$, можна перейти до моделі першого порядку.

Найбільш універсальним алгоритмом визначення параметрів моделі є метод найменших квадратів (МНК). З його допомогою можна побудувати модель не лише другого, а й вищих порядків.

Для використання МНК необхідні масиви значень вхідних та вихідних сигналів об'єкта, знятих через деякий інтервал часу T_k – період квантування. У вхідному сигналі об'єкта повинна бути як постійна, так і пробна складові. Постійна складова визначає положення робочої точки процесу, в околиці якої проводиться визначення параметрів динамічної моделі об'єкта. Таким чином, робота відбувається з цифровою (дискретною) моделлю об'єкта.

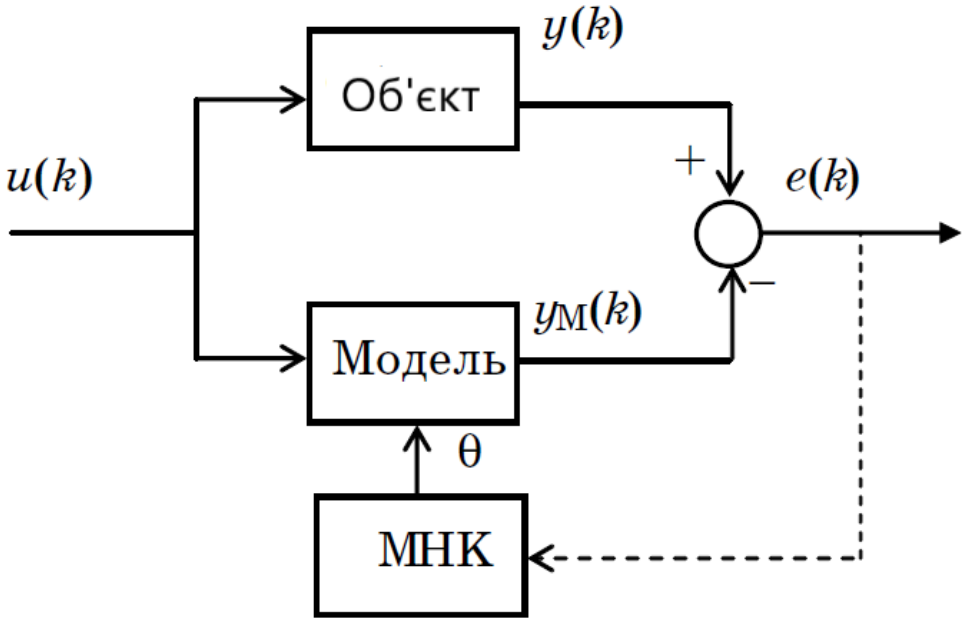
Нехай цифрова модель першого порядку задана у вигляді

$$y_M(k) = ay(k-1) + bu(k-1).$$

Схема експерименту показано на рис. 8.5.

Нехай накопичено (N + 1) точок вимірювання вхідного та вихідного сигналів об'єкта. У методі найменших квадратів узагальнена помилка ідентифікації має бути мінімальною:

$$E = \sum_{k=1}^{N+1} e(k)^2 \to \min_{\Theta}$$



$u(k)$, $y(k)$, y_M(k); θ – вхідний та вихідний сигнали об'єкта, вихідний сигнал моделі та вектор оцінки параметрів; $e(k)$ – поточна помилка ідентифікації

Рисунок 8.5 – Структурна схема експерименту

Введемо позначення

$$\mathrm{A} = \mathrm{y(k)} - \mathrm{ay(k-1)}; \mathrm{B} = \mathrm{bu(k-1)},$$

тоді

$$E = \sum_{k=1}^{N+1} [A - B]^2.$$

Після розкриття дужок та приведення подібних отримаємо

$$E = S_1 - 2aS_2 + a^2S_3 - 2bS_4 + 2abS_5 + b^2S_6,$$

де

$$S_1 = \sum y^2(k);$$
$$S_2 = \sum y(k)y(k-1);$$
$$S_3 = \sum y^2(k-1);$$
$$S_4 = \sum y(k)u(k-1);$$

$$S_5 = \sum y(k)u(k-1);$$
$$S_6 = \sum u^2(k-1).$$

Умова мінімуму функції *E* передбачає рівність нуля приватних похідних за параметрами *a* та *b*:

$$\frac{\partial E}{\partial a} = 0;\ \frac{\partial E}{\partial b} = 0;$$

$$\begin{cases} \frac{\partial E}{\partial a} = -S_2 + aS_3 + bS_5 = 0 \\ \frac{\partial E}{\partial b} = -S_4 + aS_5 + bS_6 = 0 \end{cases} \Rightarrow \begin{cases} S_2 = aS_3 + bS_5 \\ S_4 = aS_5 + bS_6 \end{cases}.$$

Тоді можна записати:

$$A\theta = B$$

Отже, для обчислення оцінок вектору параметрів об'єкта управління методом найменших квадратів отримано формулу

$$\theta = A^{-1}B.$$

Зворотна матриця A^{-1} завжди існує, оскільки вихідна матриця *A* симетрична та позитивно визначена.

Знаючи параметри дискретної моделі, можна визначити параметри передаточної функції об'єкта.

$$W(p) = \frac{K}{Tp+1} = \frac{y(p)}{u(p)}.$$

Зв'язок між параметрами дискретної моделі та передаточної функції визначається формулами:

$$a = e^{-\frac{T_k}{T}};\ b = K(1-a).$$

Звідки випливає, що

$$T = -\frac{T_k}{\ln(a)};\ K = \frac{b}{1-a}.$$

У пакеті MatLab подібні перетворення виконуються однаково для моделей будь-якого порядку.

При використанні МНК оцінки, що одержуються, обчислюються з деякими помилками, які називаються зміщенням оцінок. Для отримання хороших результатів необхідно виконати ряд умов, головні з яких полягають у тому, щоб сигнал, що тестує, був досить різноманітним, а обсяг вибірки був досить великим.

Розглянемо приклади ідентифікації.

Приклад. Дослідження поведінки об'єкта за умов обурень. Необхідно ідентифікувати постійні часу об'єкта за перехідною функцією і помилку в процесі, що встановився.

Вихідні дані:

1. Структурну схему об'єкта представлено на рис. 8.6.

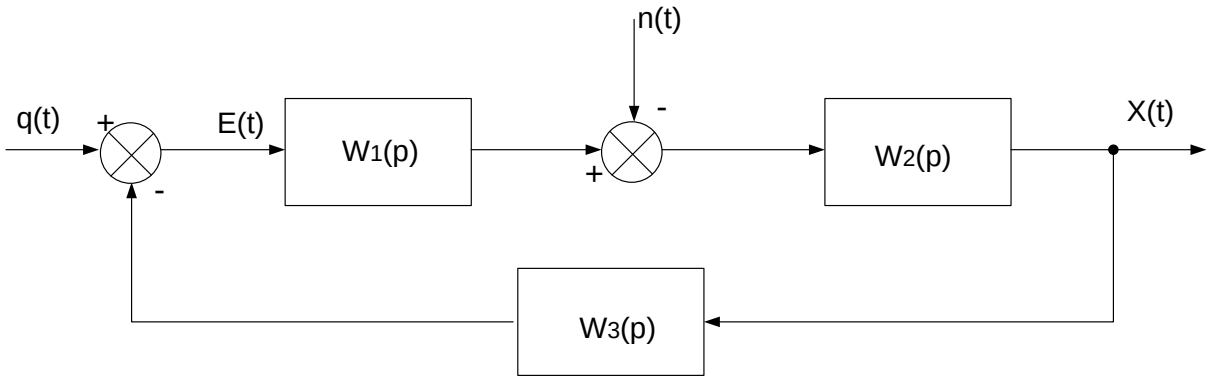


Рисунок 8.6 – Структурна схема об'єкта

2. Передаточні функції структурних блоків:

$$W_1(p)=\frac{K_1}{T_1 p+1};\quad W_2(p)=\frac{K_2}{T_2 p+1};\quad W_3=K_3.$$

3. Значення коефіцієнтів та постійних часу: K_1=5; K_2=1; K_3=0,2; T_1=0,05*с*; T_2=0,25*с*.

4. Моделювання перехідного процесу зробити із застосуванням пакета MATLAB (Simulink).

Обчислення параметрів перехідної функції виконати із застосуванням пакета MathCAD.

Рішення:

1. Визначимо передатну функцію замкнутої системи, для подальшого порівняння результатів ідентифікації:

$$W_3(p) = \frac{W_1(p)W_2(p)}{1 + W_1(p)W_2(p)W_3(p)} = \frac{\frac{5}{0.05p+1} \cdot \frac{1}{0.25p+1}}{1 + \frac{5}{0.05p+1} \cdot \frac{1}{0.25p+1} \cdot 0.2} =$$

$$= \frac{\frac{5}{(0.05p+1)(0.25p+1)}}{1 + \frac{1}{(0.05p+1)(0.25p+1)}} = \frac{5}{(0.05p+1)(0.25p+1)} \cdot$$

$$\cdot \frac{(0.05p+1)(0.25p+1)}{1 + (0.05p+1)(0.25p+1)} = \frac{5}{1 + 0.05 \cdot 0.25p^2 + 0.25p + 0.05p + 1} =$$

$$= \frac{5}{0.0125p^2 + 0.3p + 2} = \frac{2.5}{0.00625p^2 + 0.15p + 1}$$

2. Здійснюємо математичне моделювання в середовищі MATLAB. Схема математичної моделі представлена рис. 8.7.

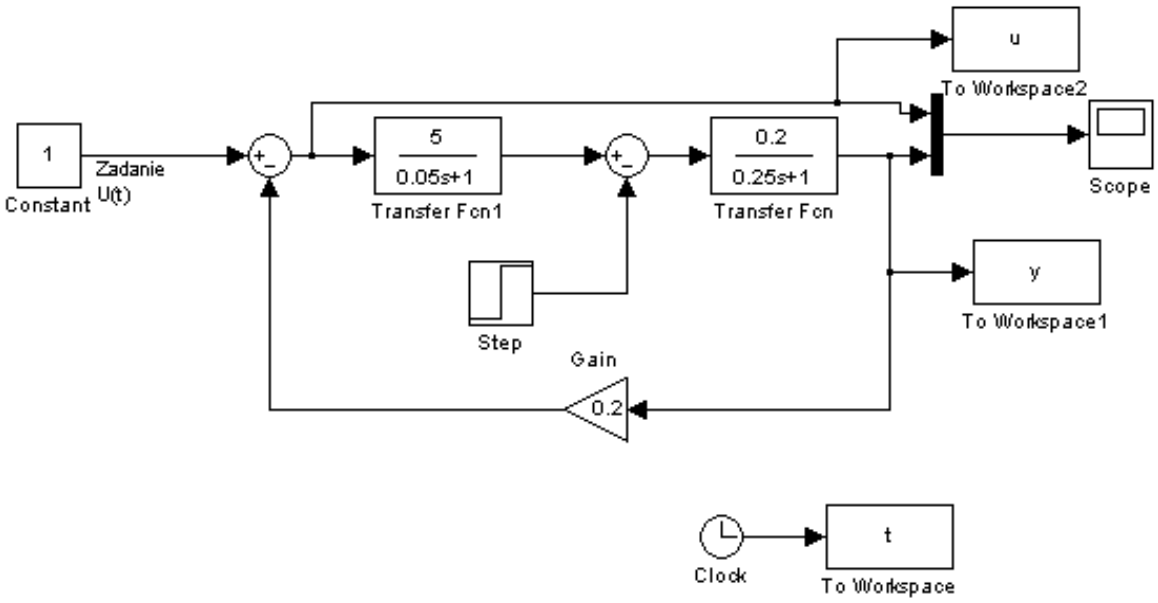


Рисунок 8.7 – Структурна схема математичної моделі об'єкта

Параметри моделювання:
- початок моделювання - 0,0
– кінець моделювання – 4,0
– спосіб моделювання – з фіксованим кроком (Fixed-Step)
– величина кроку – auto (встановлена автоматично).
Для блоку Step, що моделює зовнішнє обурення, встановлюємо

такі параметри:

– час настання перепаду сигналу – 2,0 (після закінчення перехідного процесу);

- Початкове значення сигналу - 0,0

- Кінцеве значення сигналу - 4,0 (приймаємо обурення на рівні 20% керуючого впливу:

$$1(t)\frac{20\%}{100\%}\cdot, \text{тобто } 0{,}2\cdot(1\cdot 5) = 1{,}0;$$

Для блоку *Scope* встановлюємо такі значення:

- координата Y - від 0 до 2,5 (коефіцієнт передачі системи).

Результати моделювання подано на рис. 8.8.

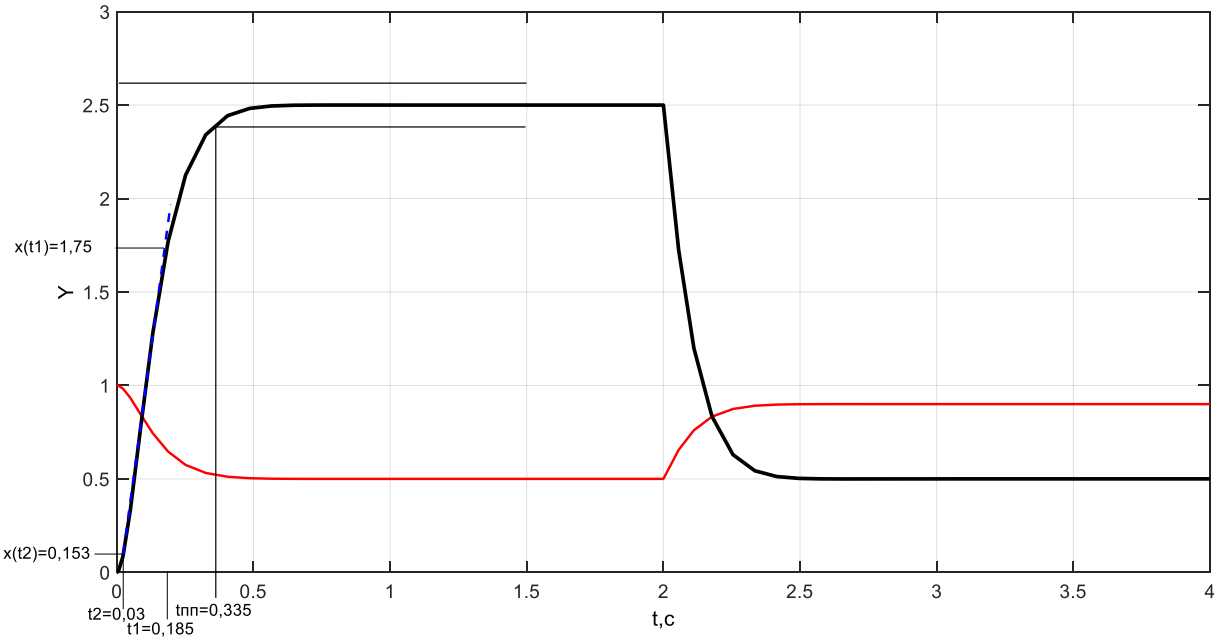


Рисунок 8.8 – Графік перехідного процесу

З графіка видно, що:

- Тривалість перехідного процесу складає $t_{пп} = 0{,}335$ *с*;

– система має аперіодичний перехідний процес, стійка;

– встановлене значення помилки $E_{вст} = 0{,}525$.

– при обуренні помилка збільшується до 0,9.

3. Постійні часу об'єкта T_1 та T_2 визначаємо із графіка перехідної функції.

Так як $T_2 > 4T_1$ і перехідний процес аперіодичний, то відносна величина вихідного сигналу $x(t)_{вітн}$ описується рівнянням:

$$x(t)_{\text{відн}} = \frac{\text{x}(t)}{x_{\text{вст}}} = 1 - 0.5\left[\frac{1+\eta}{\eta}\,\text{e}^{\frac{-2t^*}{1+\eta}} + \frac{1-\eta}{\eta}\,\text{e}^{\frac{-2t^*}{1-\eta}}\right],$$

де $t^* = \frac{t}{T_2}$ – відносний час; $\eta = \sqrt{1 - \frac{4T_1}{T_2}}$ – коефіцієнт, який характеризує відношення T_1 та T_2; $\zeta = \sqrt{\frac{4T_1}{T_2}}$ – коефіцієнт демпфірування системи; $x(t)_{отн} = 0...1$.

Визначаємо коефіцієнт, що характеризує співвідношення постійних часу.

$$\eta = \sqrt{1 - \frac{4T_1}{T_2}} = \sqrt{1 - \frac{4 \cdot 0{,}05}{0{,}25}} = 0{,}447.$$

Використовуючи відомий висновок, що величина η практично не впливає на β при $x(t)_{отн} = 0{,}7$, для якої $t^*_{0,7} \approx 1{,}2$.

Визначаємо значення $x(t_1)$ при $x(t)_{отн} = 0{,}7$:

$$x(t_1) = 0{,}7 x_{уст} = 0{,}7 \cdot 2{,}5 = 1{,}75.$$

З графіка перехідного процесу (див. рис. 11.8) визначаємо час встановлення значення $x(t_1) = 1{,}75$, яке дорівнює $t_{0,7} = 0{,}185$.

Так як $t^*_{0,7} = 1{,}2$, постійна асу T_2 для $t_{0,7} = 0{,}185$ становить:

$$T_2 = \frac{t_{0,7}}{t^*_{0,7}} = \frac{0{,}185}{1{,}2} = 0{,}154.$$

Задаючи інший відносний час $t^* = 0.2$ знаходим

$$t_2 = 0{,}2 \cdot T_2 = 0{,}2 \cdot 0{,}154 = 0{,}0308.$$

З графіка перехідного процесу визначається $x(t_2) = 0{,}153$

$$x_{отн}(t_2) = \frac{0{,}153}{2{,}5} = 0{,}0617.$$

4. Використовуючи середовище математичної прикладної програми *MathCAD*, за допомогою функції *Given* розв'язуємо рівняння $x(t)$ щодо η:

$$XX := 0.0617$$

$$\text{Given} \qquad \eta := 0.477$$

$$1 - \left(\frac{1+\eta}{2\cdot\eta} \cdot e^{\frac{-2\cdot 0.4}{1+\eta}} \right) - \left(\frac{1-\eta}{\eta\cdot 2} \cdot e^{\frac{-2\cdot 0.4}{1-\eta}} \right) = XX$$

$$\eta 2 := \text{Find}(\eta) \qquad \eta 2 = 0.529$$

$$\eta := 0.526$$

$$\hat{T}_1 = \frac{1}{4} \cdot (1-\eta^2) \cdot \hat{T}_2 = \frac{1}{4} \cdot (1-0{,}526^2) \cdot 0{,}154 = 0{,}028\ c.$$

Визначимо помилку ідентифікації:

$$\Delta T_1 = \hat{T}_{z1} - T_1 = 0.05 - 0.028 = 0.022;$$
$$\Delta T_2 = \hat{T}_{z2} - T_2 = 0.25 - 0.154 = 0.096.$$

Визначаємо умову аперіодичного процесу

$$\xi = \frac{T_2}{4\cdot T_1} \geq 1, \qquad \xi = \frac{0{,}154}{4\cdot 0{,}028} = 1.38 \rangle 1.$$

Висновок: Постійні часу T_1 та T_2 можуть бути ідентифіковані з достатньою точністю за графіком аперіодичного перехідного процесу (у тимчасовій області).

Приклад. Необхідно дослідити поведінку об'єкта та ідентифікувати його параметри у частотній області

Вихідні дані:

1. Передатна функція об'єкта:

$$W_{0(p)} = \frac{k}{a_0 p^2 + a_1 p + 1}.$$

2. Значення коефіцієнту передачі та постійних часу: $k = 2$, $a_0 = 0{,}2$, $a_1 = 0{,}3$.

Рішення:

1. Складаємо структурну схему дослідження моделі з використанням пакета *Simulink* у середовищі *MATLAB* (рис. 8.9).

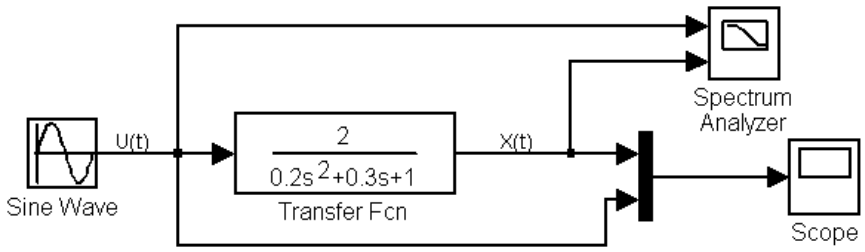


Рисунок 8.9 – Схема дослідження математичної моделі у пакеті MatLab Simulink

Параметри налаштування моделі:
- Час моделювання - 0,0 ÷ 6,0 с;
- Амплітуда синусоїдального сигналу - 1,0;
- Постійна складова (Bias) - 0,0;
- Частота (Frequency) - 1,0; 5,0; 10,0 с-1;
- Мультиплексор - 2 входи, спосіб відображення - bar;
- Осцилограф - координата *Y* - від -15 до 15;

Результати досліджень ідеальної та реальної перехідної характеристики, а також амплітудно-частотної та фазо-частотної характеристик вихідного сигналу при різних частотах керуючого сигналу наведено на рис. 8.10–8.12.

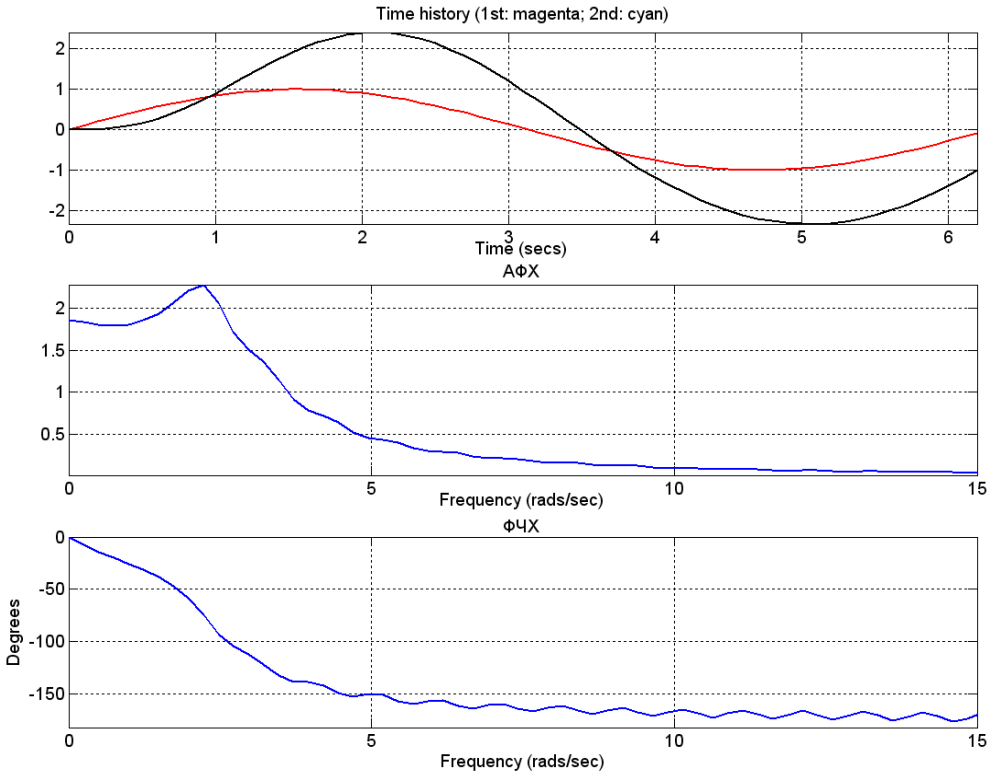


Рисунок 8.10 – Ідеальна та реальна перехідна, амплітудно-частотна та фазо-частотна характеристики вихідного сигналу при частоті $\omega = 1c^{-1}$

Обробка результатів математичного моделювання проводиться у наступній послідовності.

Вхідний сигнал описується виразом

$$U_{i(t)} = 1 \cdot \sin \omega_i t \,,$$

де $\omega_i \in [1;5;10] c^{-1}$.

Вихідний сигнал *X(t)*:

$$X_{i(t)} = A_i \, sin[\, \omega_i t + \phi_{(\omega_i)}].$$

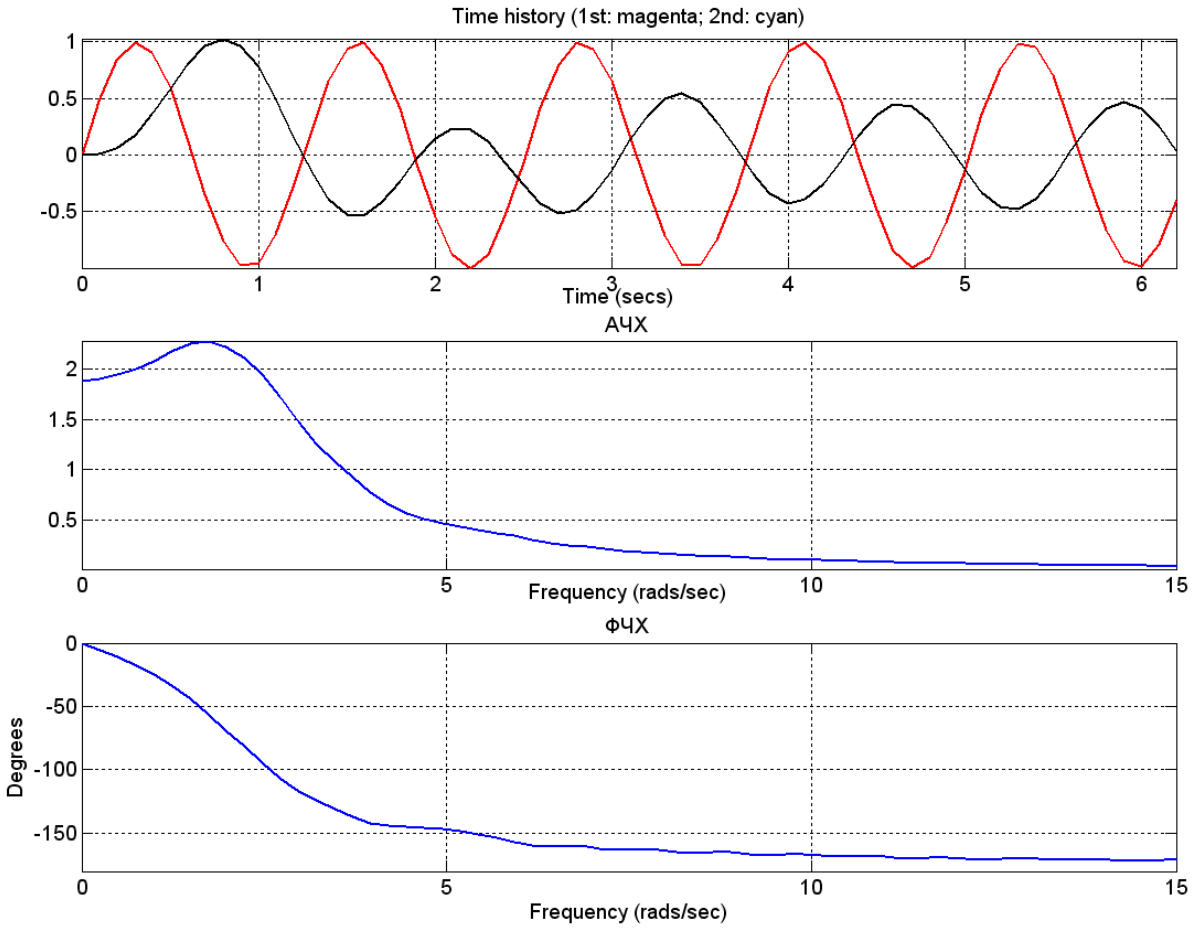


Рисунок 8.11 – Ідеальна та реальна перехідна, амплітудно-частотна та фазо-частотна характеристики вихідного сигналу при частоті $\omega = 5c^{-1}$

Відношення цих сигналів визначається передаточною функцією об'єкта:

$$W_0(p) = \frac{X(t)}{U(t)} = \frac{2}{0{,}2p^2 + 0{,}3p + 1}$$

У загальному випадку передатна функція при переході до

полярних координат (амплітудно-фазової характеристики) описується ставленням комплексних функцій чисельника (R) і знаменника (Q) або подається в комплексній формі наступним чином:

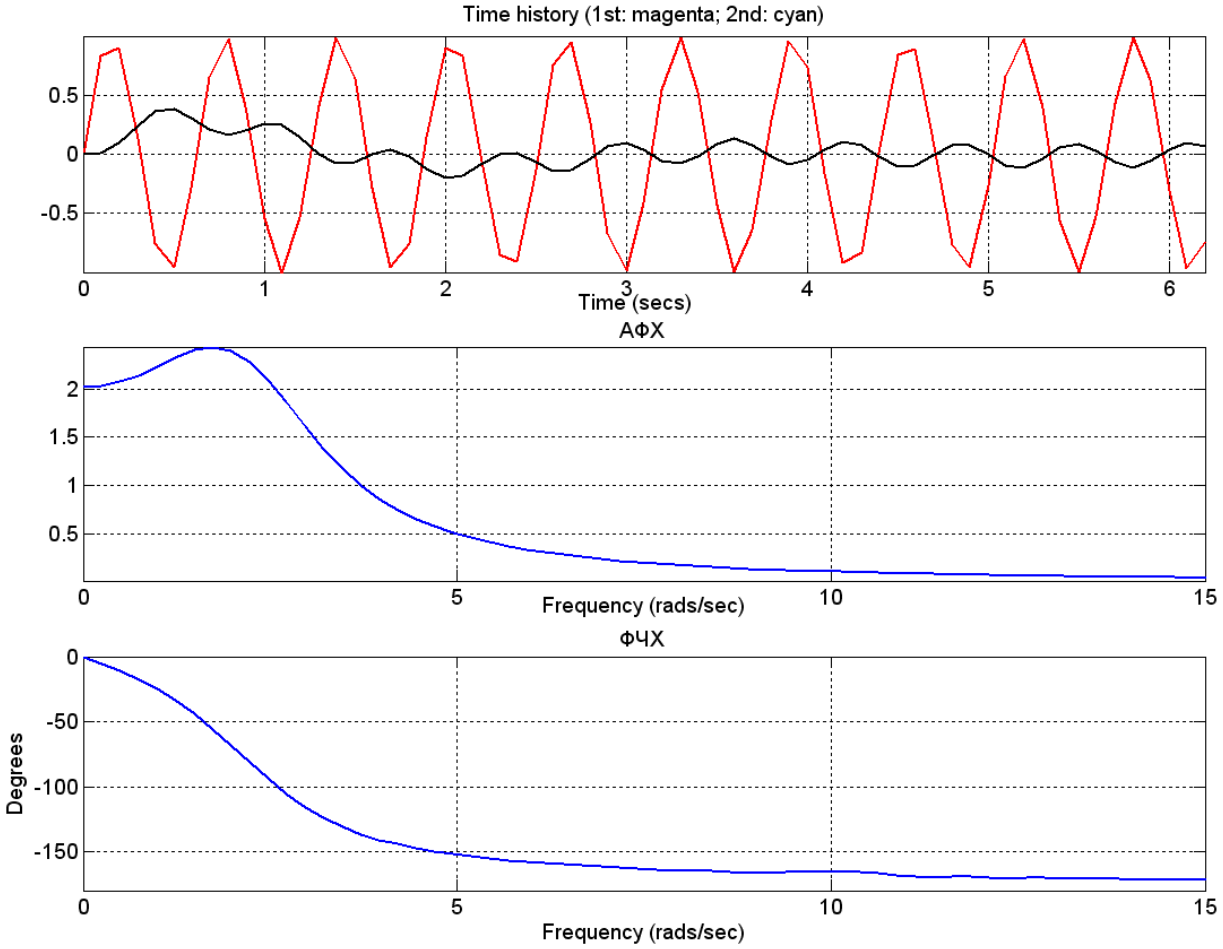


Рисунок 8.12 – Ідеальна та реальна перехідна, амплітудно-частотна та фазо-частотна характеристики вихідного сигналу при частоті $\omega = 10c^{-1}$

$$W_0(j\omega) = \frac{U_R(\omega) + jV_R(\omega)}{U_Q(\omega) + jV_Q(\omega)} = U(\omega) + jV(\omega)$$

Звідси чисельник:

$$U_R(\omega) + jV_R(\omega) = U(\omega)U_Q(\omega) + jV_Q(\omega)U(\omega) + U_Q(\omega)jV(\omega) - V_Q(\omega)V(\omega).$$

Після групування речовинної та уявної складових отримаємо:

$$\begin{cases} U_R(\omega) = U(\omega)U_Q(\omega) - V_Q(\omega)V(\omega), \\ V_R(\omega) = U(\omega)V_Q(\omega) + U_Q(\omega)V(\omega) \end{cases}$$

З графіків на рис. 8.10–8.12 визначаємо:

1. ω_1= 1 c^{-1}, A_1=1,83, φ_1= –25,9;
2. ω_2= 5 c^{-1}, A_2=0,46, φ_2= –147;
3. ω_3= 10 c^{-1}, A_3=0,105, φ_3= –164,5.

Визначаємо речові та уявні характеристики об'єкта в цих точках:

1. ω_1= 1 c^{-1}:

$$U_1 = A_1\, cos\left(\phi_1 \cdot \frac{\pi}{180}\right) = 1{,}83\, cos\left(-25.9\frac{\pi}{180}\right) = 1.646,$$
$$V_1 = A_1\, sin\left(\phi_1 \cdot \frac{\pi}{180}\right) = 1.83 \cdot sin\left(-25.9\frac{\pi}{180}\right) = -0.799.$$

2. ω_2= 5 c^{-1}:

$$U_2 = A_2\, cos\left(\phi_2 \cdot \frac{\pi}{180}\right) = 0.46\, cos\left(-147\frac{\pi}{180}\right) = -0.386,$$
$$V_2 = A_2\, sin\left(\phi_2 \cdot \frac{\pi}{180}\right) = 0.46 \cdot sin\left(-147\frac{\pi}{180}\right) = -0.251.$$

3. ω_3= 10 c^{-1}:

$$U_3 = A_3\, cos\left(\phi_3 \cdot \frac{\pi}{180}\right) = 0.105\, cos\left(-164.5\frac{\pi}{180}\right) = -0.101,$$
$$V_3 = A_3\, sin\left(\phi_3 \cdot \frac{\pi}{180}\right) = 0.105 \cdot sin\left(-164.5\frac{\pi}{180}\right) = -0.028.$$

Визначаємо речові та уявні характеристики чисельника та знаменника АФХ об'єкта:

$$W_0(j\omega) = \frac{k}{a_0(j\omega)^2 + a_1(j\omega) + 1} = \frac{U_R(\omega)}{U_Q(\omega) + jV_Q(\omega)}.$$

де:

$$U_R(\omega) = k;$$
$$jV_R(\omega) = 0;$$
$$U_Q(\omega) = -a_0\omega^2 + 1;$$
$$jV_Q(\omega) = j(a_1\omega).$$

Складаємо систему рівнянь для кожного із трьох експериментальних результатів:

ω_1= 1 c^{-1}:

$$k = 1.646 \cdot (1 - a_0 \cdot 1^2) - a_1 \cdot 1 \cdot (-0.799);$$
$$0 = 1.646 \cdot a_1 \cdot 1 + (1 - a_0 \cdot 1^2) \cdot (-0.799).$$

ω_2= 5 c^{-1}:

$$k = (-0.386) \cdot (1 - a_0 \cdot 5^2) - a_1 \cdot 5 \cdot (-0{,}251);$$
$$0 = (-0{,}386) \cdot a_1 \cdot 5 + (1 - a_0 \cdot 5^2) \cdot (-0{,}251).$$

ω_3= 10 c^{-1}:

$$k = (-0{,}101) \cdot (1 - a_0 \cdot 10^2) - a_1 \cdot 10 \cdot (-0{,}028);$$
$$0 = (-0{,}101) \cdot a_1 \cdot 10 + (1 - a_0 \cdot 10^2) \cdot (-0{,}028).$$

Із застосуванням пакету MathCad визначаємо рішення системи рівнянь при ω_1= 1 c^{-1} та ω_3= 10 c^{-1} відностно k, a_0, a_1:

Given

$$U1 \cdot a1 \cdot \omega1 + V1 \cdot \left[1 - a0 \cdot (\omega1)^2\right] = 0$$

$$U3 \cdot a1 \cdot \omega3 + V3 \cdot \left[1 - a0 \cdot (\omega3)^2\right] = 0$$

$a := \text{Find}(a0, a1)$

$a_0 = 0.158$

$a_1 = 0.409$

$k := U1 \cdot \left(1 - a_0 \cdot \omega1^2\right) - V1 \cdot a_1 \cdot \omega1$ $\quad k = 1.714$

$k := U2 \cdot \left(1 - a_0 \cdot \omega2^2\right) - V2 \cdot a_1 \cdot \omega2$ $\quad k = 1.646$

$k := U3 \cdot \left(1 - a_0 \cdot \omega3^2\right) - V3 \cdot a_1 \cdot \omega3$ $\quad k = 1.607$

Для побудови амплітудно-фазової частотної характеристики у командному вікні середовища MatLab скористаємося командою nyquist(sys), а саме

```
H=tf([2],[0.2 0.3 1])
nyquist(H)
```

Результати виконання команди наведені на рисунку 8.13

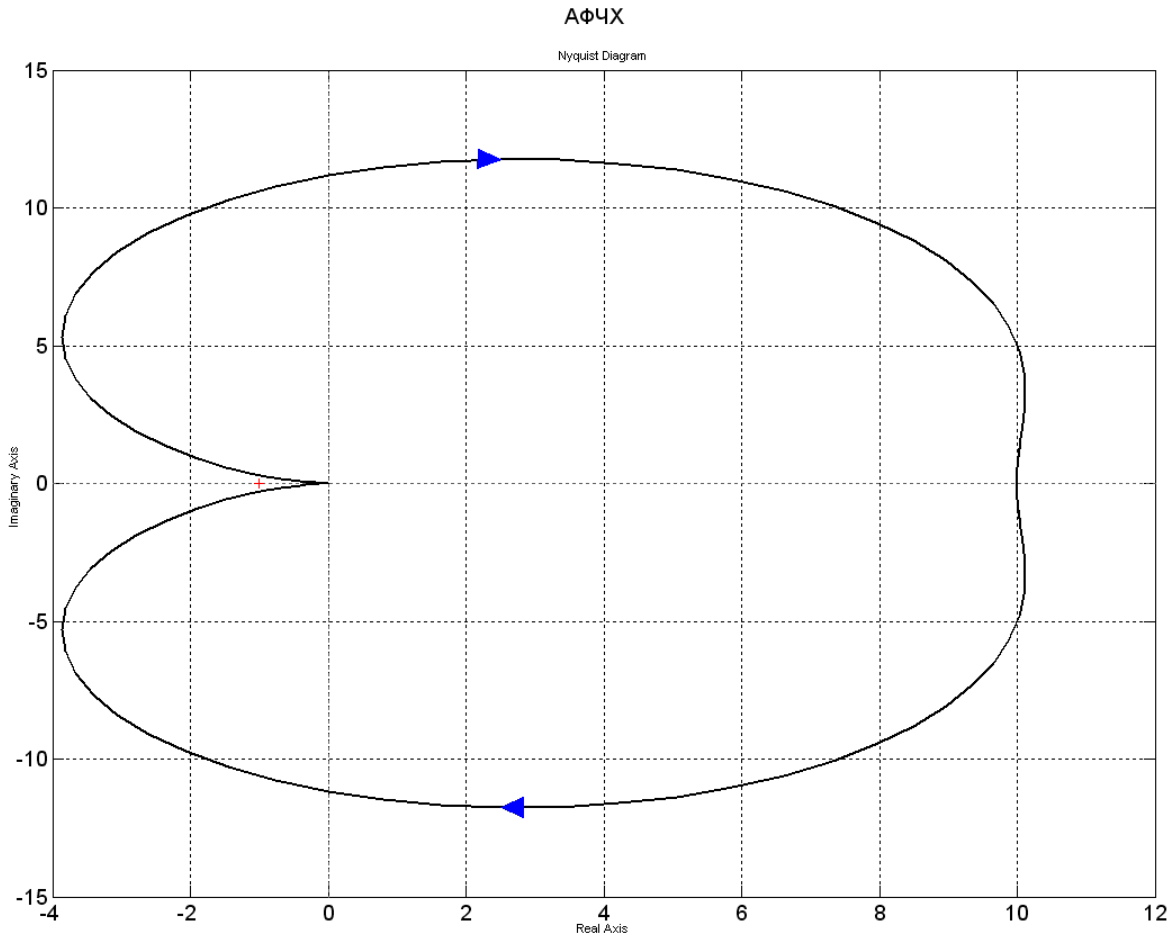


Рисунок 8.13 – Амплітудно-фазова частотна характеристика математичної моделі

8.3 Ідентифікація динамічних ланок нейронною мережею

У процесі ідентифікації потрібно за відомими вхідно-вихідними залежностями динамічного об'єкта побудувати його опис, який можна використовувати для передбачення вихідного сигналу при довільному вхідному.

Проте, як зазначалося, ШНМ ПП – це не динамічна мережа. Простий шлях внесення динаміки в поведінку ШНМ полягає у подачі на вхід мережі не тільки поточних, а й затриманих значень входу та виходу. Число затриманих сигналів і величина затримки залежить від конкретного об'єкта.

На рисунку 8.14 наведено принцип використання нейромережевої моделі як ідентифікації. Число ліній затримки Δ на входах ШНМ повинно приблизно відповідати порядку об'єкта. У цій схемі за допомогою алгоритму зворотного розповсюдження має мінімізуватися помилка між виходом об'єкта $y(t)$ та виходом моделі $y^m(t)$.

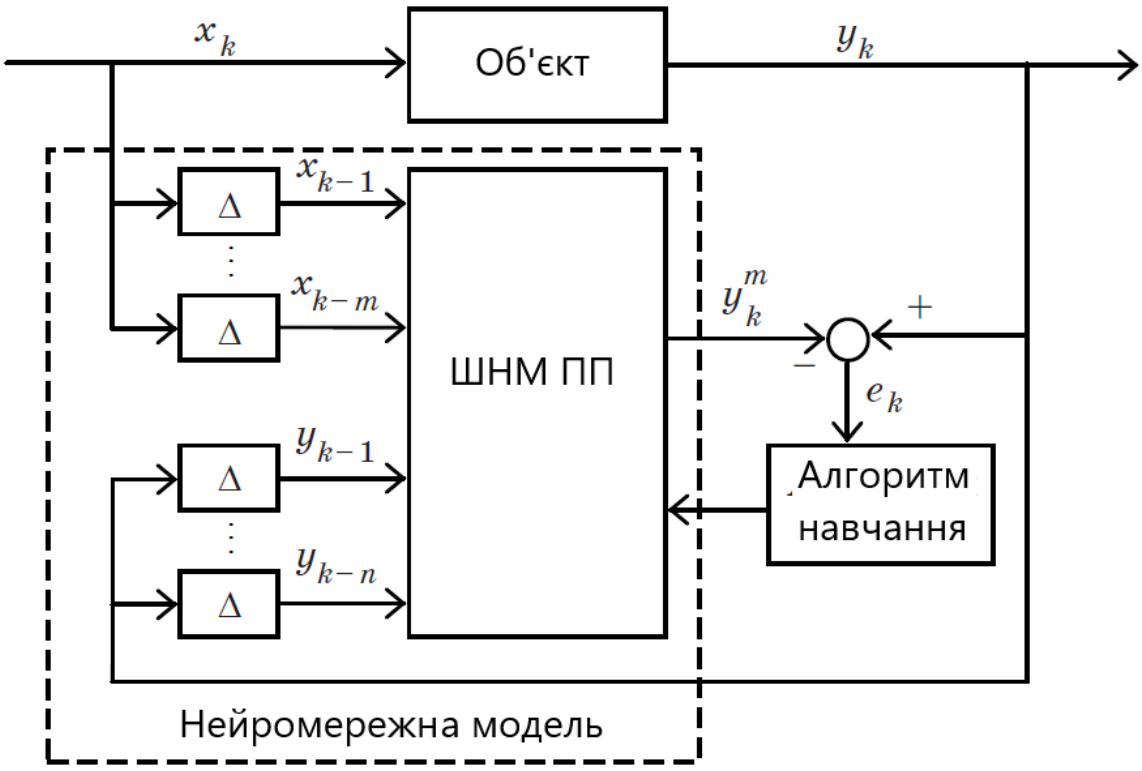


Рисунок 8.14 – Нейромережева ідентифікація

Отримана нейромережева модель є «чорною скринькою». Вона не дозволяє судити про фізичні процеси, що протікають в об'єкті управління, але може бути ефективно використана для аналізу та прогнозу поведінки об'єкта, а також для синтезу системи управління.

Для лінійних і слабонелінійних об'єктів управління класичні методи ідентифікації можуть не поступатися нейромережевим методам. Однак ШНМ є універсальним інструментом і придатна для ідентифікації суттєво нелінійних об'єктів, про які є малий обсяг апріорної інформації.

Приклад 8.1. Розглянемо лінійний динамічний об'єкт із прикладу 4.11 (перехідний процес наведено на рис. 4.23).

$$\frac{d^2y(t)}{dt^2} + 0{,}5\frac{dy(t)}{dt} + y(t) = x(t) \Leftrightarrow W(s) = \frac{Y(s)}{X(s)} = \frac{1}{s^2 + 0.5s + 1}.$$

\Це об'єкт 2-го порядку, тому йому може відповідати ШНМ із двома лініями затримки (див. рис. 8.15).

Зауважимо, що структурна надмірність є характерною властивістю ШНМ, тому в структурі, наведеній на рис. 5.2 можна було вибрати не три, а більше нейронів прихованого шару.

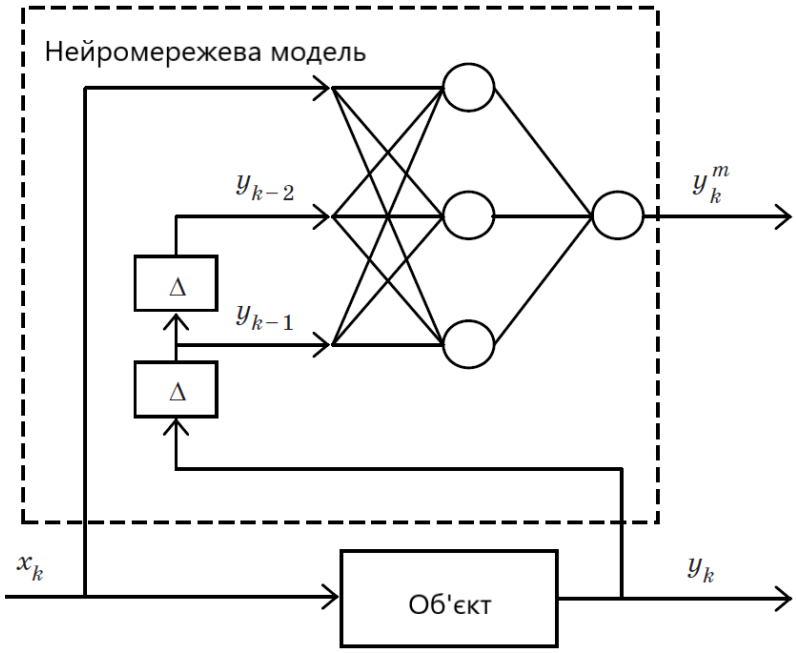


Рисунок 8.15 - Нейромережева ідентифікація об'єкта 2-го порядку

Схема моделювання представлена на рис. 8.16 (використовується метод інтегрування з постійним кроком 0,01). Виконаємо моделювання (примітка: PR51 - ім'я файлу моделі, 20 – час моделювання):

```
simOut=sim('PR51',20);
figure(1);
plot(simOut.t,simOut.Y,simOut.t,simOut.Y1);
xlabel('t,c');
ylabel('Y(t)');
grid;
net=newff([0 1; -3 3; -3 3], [3,1], {'purelin','purelin'},'trainlm');
P = simOut.Y';
T = simOut.Y1';
net.trainParam.show = 50;
net.trainParam.lr = 0.005;
net.trainParam.epochs = 1000;
net.trainParam.goal = 0.0001;
net1 = train(net, P, T);
```

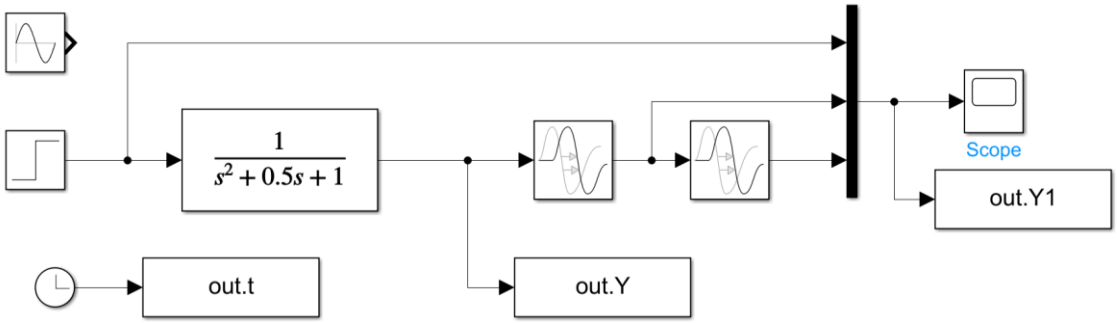


Рисунок 8.16 - Схема підготовки даних для ідентифікації

Для навчання знадобилося лише 5 епох (рис. 8.17).

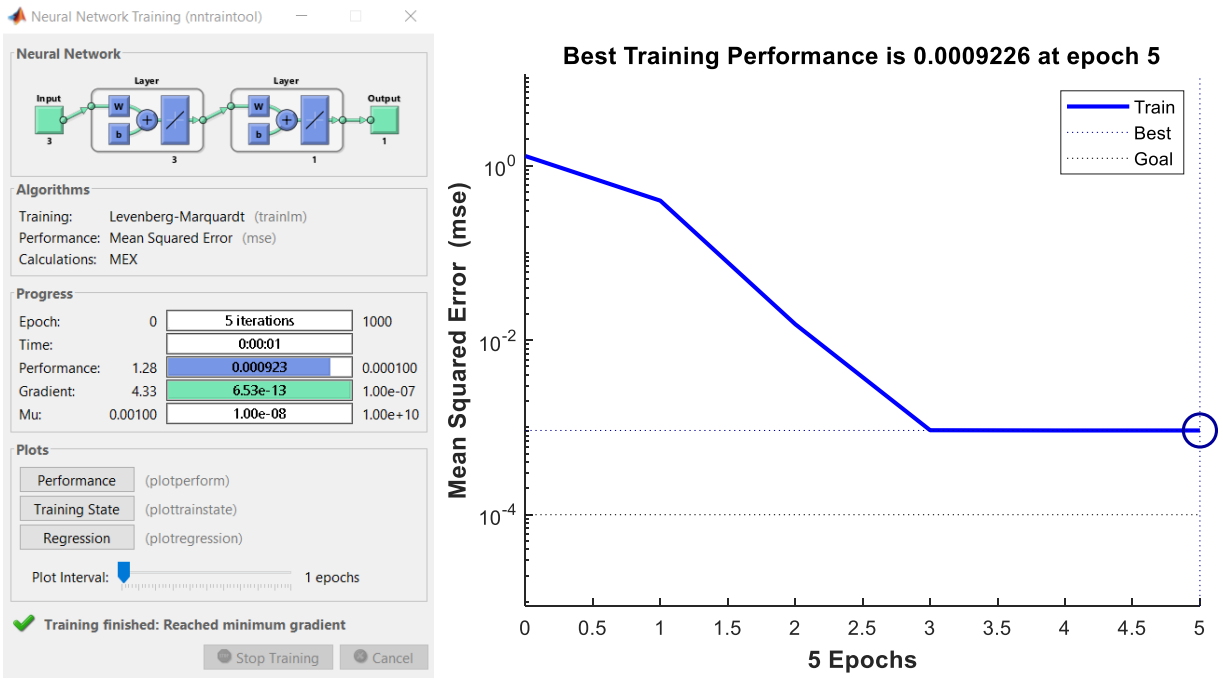


Рисунок 8.17 – Процес навчання нейромережевого ідентифікатора

Виконаємо перевірку нейромережевої моделі структурна схема якої наведена на рис. 8.19. За допомогою наступної команди створюється блок Neural Network у Simulink-моделі:

>> gensim(net1,0.01)
де 0,01 - крок інтегрування за часом.

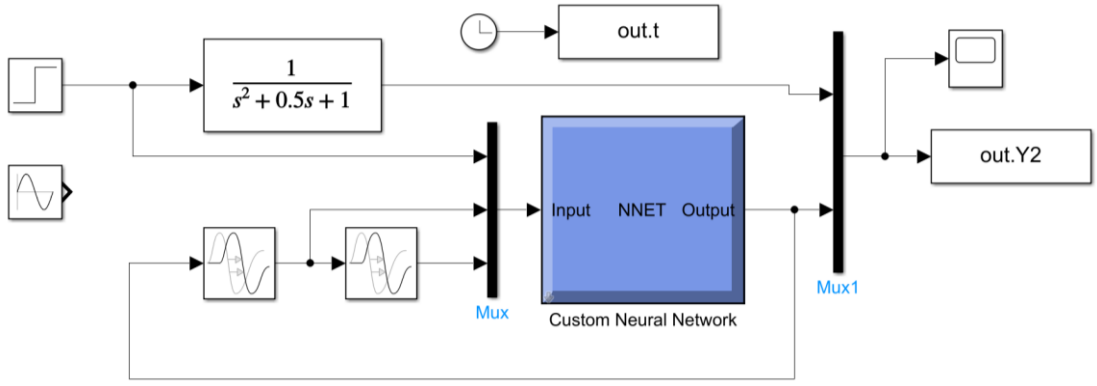


Рисунок 8.19 - Схема перевірки якості нейромережевої моделі

З аналізу графіків перехідного процесу на рис. 8.20 отриманих при моделювання математичної моделі наведеної на рисунку 8.19, можливо зробити висновок, що виходи об'єкта та нейромережевої моделі досить близькі. Підвищення якості ідентифікації можна досягти, змінюючи параметри нейромережевої моделі

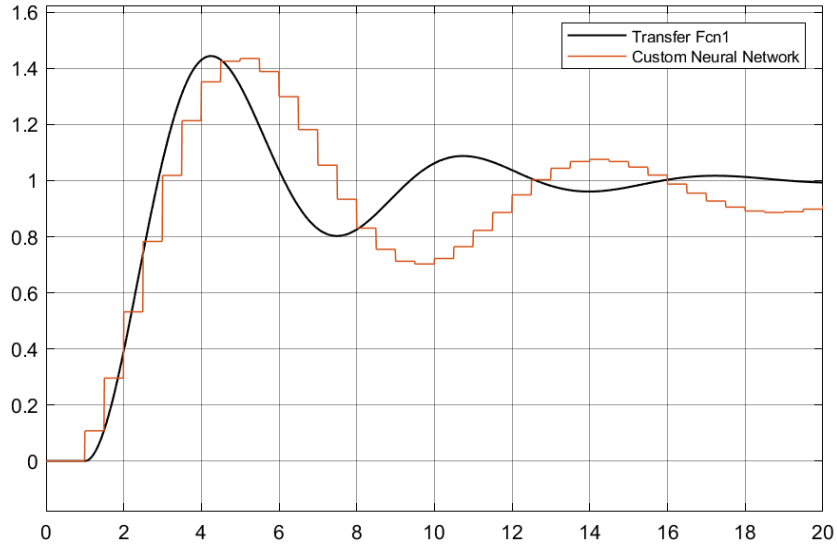


Рисунок 8.20 – Порівняльні графіки перехідного процесу об'єкту та його нейромережевої моделі

Приклад 8.2. Розглянемо приклад ідентифікації нелінійної системи:

$$y(k+1) = \frac{y(k)(y(k-1)+2)(y(k)+3)}{8+y(k)^2+y(k-1)^2} + u(k).$$

Перепишем рівняння системи в формі:

$$y(k+1) = f(y(k), y(k-1)) + u(k).$$

Тоді завдання ідентифікації можна подати у вигляді, наведеному на рис. 8.20 де нейронна мережа реалізує оператор *f*.

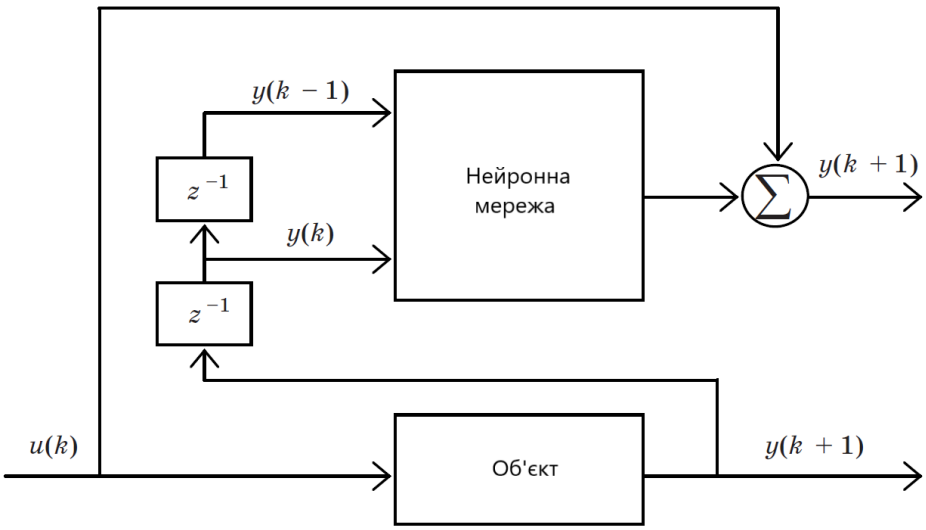


Рисунок 8.20 – Ідентифікація нелінійного об'єкта

Опишемо вхідний сигнал (система стійка при u ∈ [–2, 2]):

```
y=rands(1, 301)*2;
u=rands(1, 301)*2;
```

Реакція системи на вхідний сигнал:

```
for k=2 : 301
y1(k+1)=((y(k)*(y(k-1) + 2)*(y(k) + 3))/(8 + y(k)^2 + y(k-1)^2));
y(k+1)=y1(k+1)+ u(k);
out(k-1)=(y(k+1)-u(k))/20;
in(k-1)=y(k)/20;
end;
```

Останні дві команди циклу нормують входи та виходи ШНМ. Потім створимо ШНМ ПП із двома входами та одним виходом:

```
net = newff([min(in) max(in); min(in) max(in)],[2 10 1],{'tansig'
'tansig' 'tansig'}, 'trainlm', 'learngdm', 'mse');
```

Сформуємо навчальні дані:

```
plantin=[in(1:299); in(2:300)];
plantout=out(1:299);
```

Навчимо нейронну мережу:

```
net.trainParam.epochs = 500;
net.trainParam.goal = 0.0005;
net = train (net, plantin, plantout);
```

Далі виконаємо перевірку навченої нейронної мережі, подавши послідовно два синусоїдальні сигнали:

```
yp(1)=0.0; yp(2)=0.0;out1(1)=0; out1(2)=0;
 for k=2:500
if (k<=200)u(k)=2.0*cos(2*pi*k*0.01);
else
u(k)=1.2*sin(2*pi*k*0.05);
end;
yp(k + 1)=yp(k)*(yp(k-1) + 2)*(yp(k) + 2.5)/(8.5 + yp(k)^2 + yp(k-1)^2) +
u(k);
out1(k)=yp(k)/20;
out1(k-1)=yp(k-1)/20;
nnout(k + 1)=20*sim(net,[out1(k);out1(k-1)]) + u(k);
end;
plot(yp, 'b');
hold on;
plot(nnout, ':k');
axis([0, 500, -4.0, 10.0]);
```

```
ylabel('StepPlant (solid) NN Output (dotted')
xlabel('Time Step')
grid
```

На рисунку 8.21 наведено графіки виходу об'єкта (суцільна лінія) та ідентифікаційної моделі (пунктир).

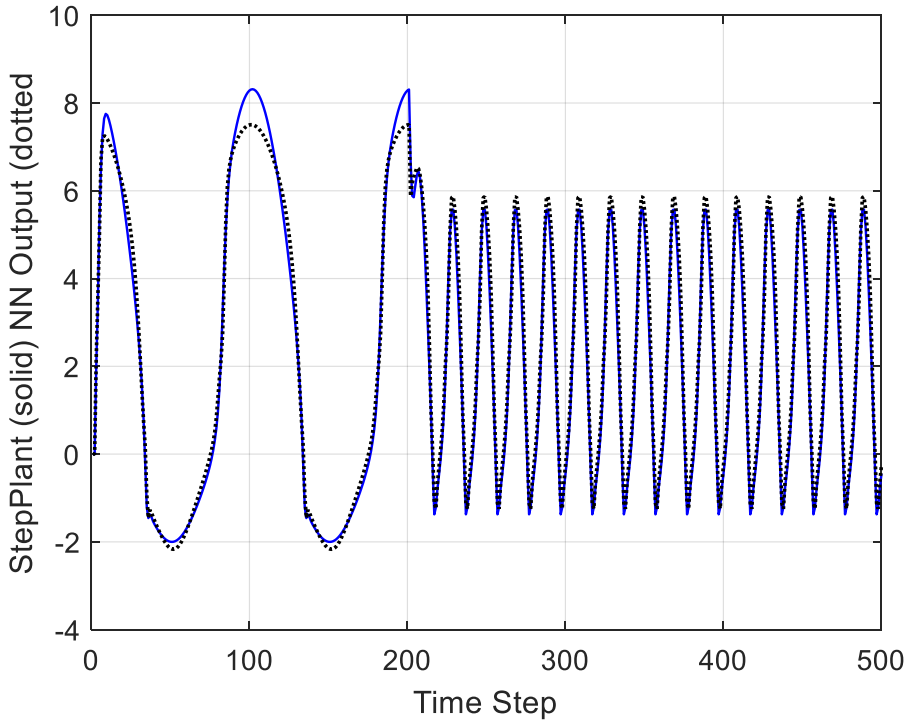


Рисунок 8.21 – Перевірка якості ідентифікації

Для навчання знадобилося 14 епох (рис. 8.22).

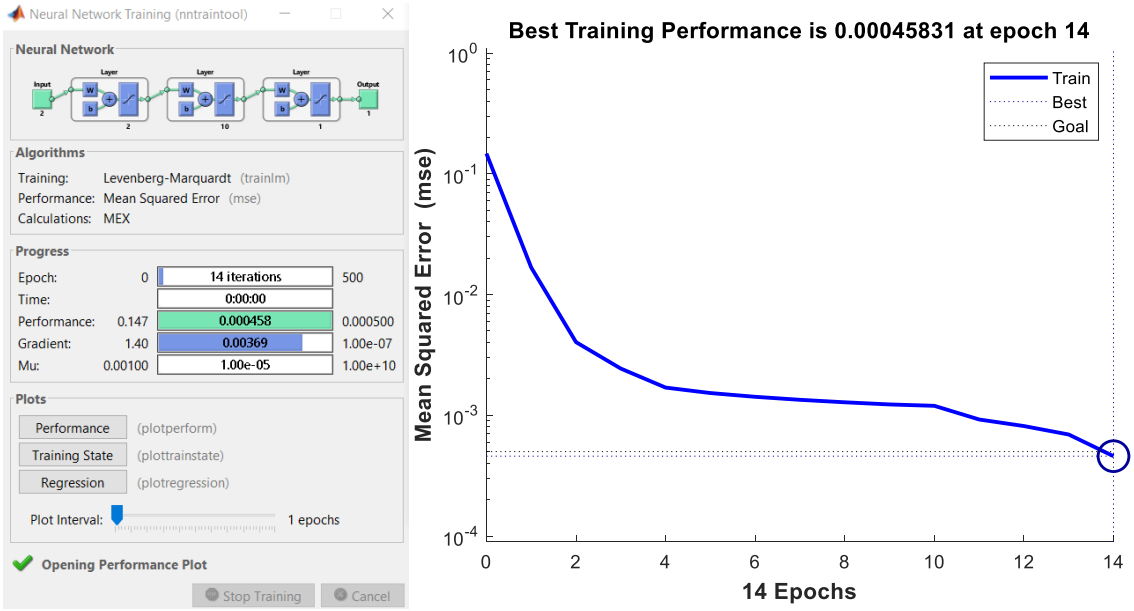


Рисунок 8.22 – Процес навчання нейромережевого ідентифікатора не лінійного об’єкту

За допомогою ШНМ можна ідентифікувати не тільки об'єкт управління, а й регулятор об'єкта, оскільки останній також є динамічною ланкою. Штучні нейронні мережі можуть бути використані у різних конфігураціях у контексті управління. Така мережа може замінювати людину в контурі управління, коли робота відбувається в небезпечних умовах або для виключення помилок через втому, недосвідченість тощо.

Для заміни існуючого регулятора (або людини-оператора) нейромережевою моделлю в режимі on line (див. рис. 8.23) слід у певні моменти часу зчитувати з датчиків сигнали з входу регулятора (тобто помилка *e*(*t*) або опис стану об'єкта *x*(*t*))) і з його виходу - сигнал керування *u*(*t*).

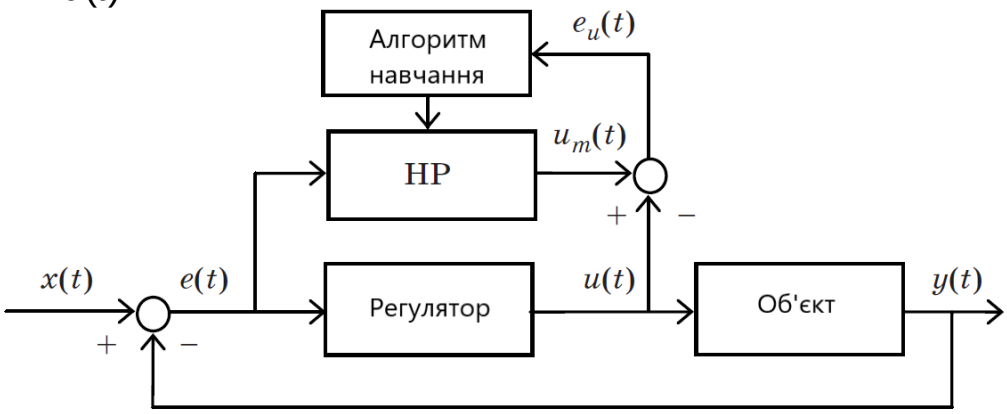


Рисунок 8.23 – Навчання з допомогою існуючого регулятора

Опис стану подається на вхід ШНМ, а опис сигналу управління використовується для розрахунку поточної помилки виходу ШНМ. Алгоритм зворотного розповсюдження помилки дозволяє мінімізувати її:

$$e_u(t) = u_m(t) - u(t).$$

При навчанні в режимі offline за допомогою існуючого регулятора створюється навчальна вибірка, і ШНМ навчається за цією вибіркою.

Приклад 8.3. Нехай є система керування з ПД-регулятором. Потрібно замінити його еквівалентним нейроном.

На рисунку 8.24 наведено вихідну схему математичної моделі, необхідну для отримання навчальної вибірки.

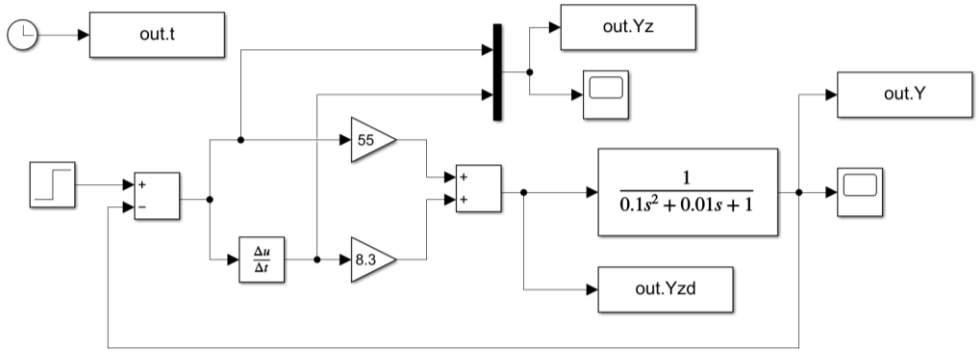


Рисунок 8.24 - Система з ПД-регулятором в MatLab Simulink

Оскільки ПД-регулятор є лінійним, для його заміни можна використовувати ШНМ ПП з лінійною активаційною функцією:

```
simOut=sim('PR53',1); % файл моделі PR53.slx (рис.8.24), 1 - час моделювання (1с)
net=newff([-1 1; -5 5], [3,1], {'purelin','purelin'},'trainlm');
P = simOut.Yz';
T = simOut.Yzd';
net.trainParam.show = 50;
net.trainParam.lr = 0.005;
net.trainParam.epochs = 1000;
net.trainParam.goal = 0.0001;
net1 = train(net, P, T);
gensim(net1,0.01)
```

Як випливає з рис. 8.25, для навчання знадобилося всього п’ять епох

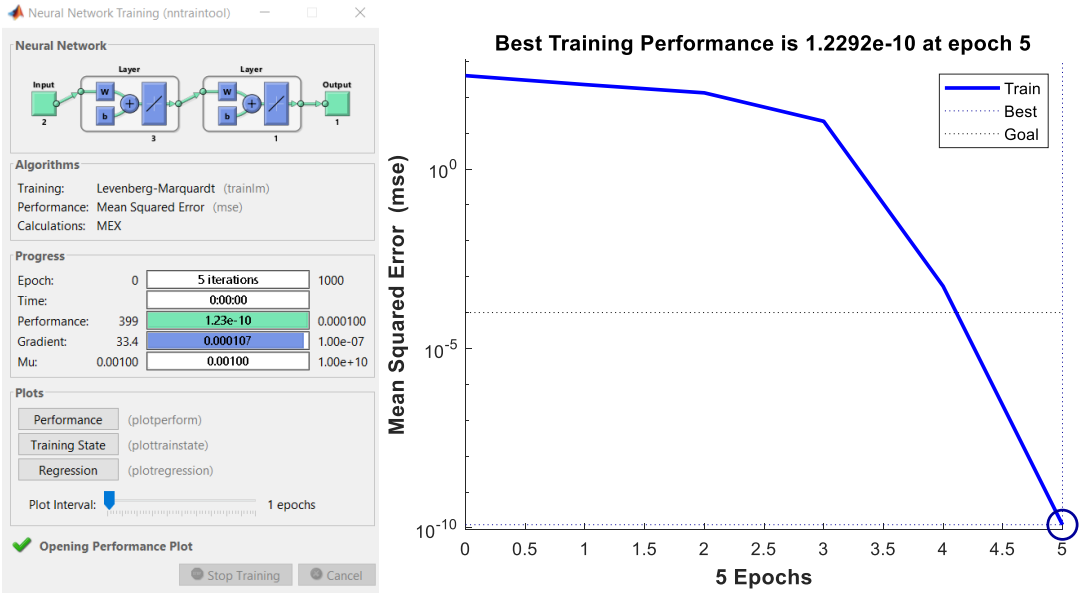


Рисунок 8.25 - Зміна помилки у процесі навчання нейронного регулятора

На рис. 8.26 наведено еквівалентну схему з нейронним регулятором.

Проведемо дослідження графіків перехідних процесів при використанні вихідного (класичного) ПД-регулятора та його нейронної моделі

```
%'Перехідні процеси'
plot(out.t,out.Y,out.t,out.Ynr);
xlabel('Time, c');
ylabel('Y(t)');
```

```
legend('Класичний регулятор','Нейрорегулятор');
grid
```

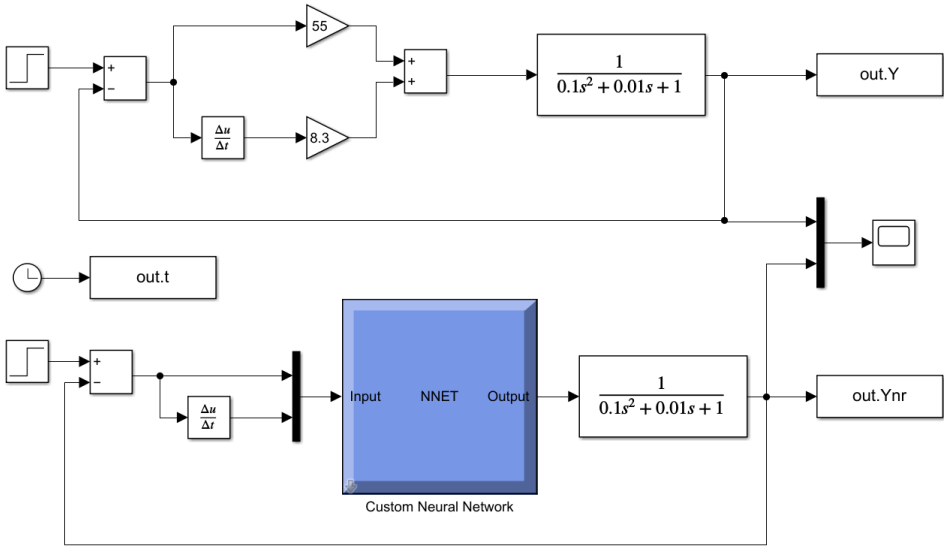


Рисунок 8.26 - Нейронний регулятор, еквівалентний ПД-регулятору

Побудовані графіки перехідних процесів наведено рисунку 8.27 . Аналіз графіків показує їхній практичний збіг.

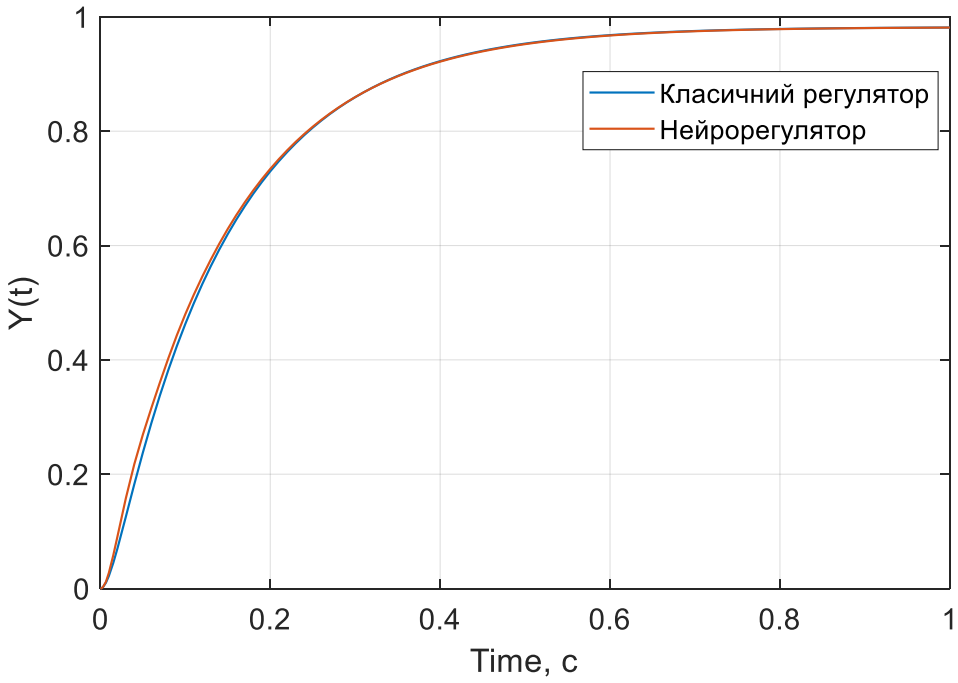


Рисунок 8.27 - Порівняння графіків перехідних процесів

Отриманий таким способом нейронний регулятор, природно, не може працювати краще за свій прототип, використаний при навчанні. Для підвищення якості керування необхідно застосовувати інші підходи.

8.4. Нейроемулятори та нейропредиктори

Подання динамічного об'єкта з урахуванням ШНМ докорінно відрізняється від опису динамічного об'єкта з допомогою диференціальних рівнянь. Аналітичний опис з допомогою диференціальних рівнянь використовує «глибинні» знання, отримані з урахуванням законів фізики, хімії, біології та інших фундаментальних наук. Нейромережеве моделювання розглядає об'єкт як «чорний ящик», у якого відомі входи та виходи. Завдання ШНМ – найкраще наслідувати поведінку цієї «чорної скриньки».

Разом з тим, при аналітичному описі використовується деякий шаблон – рівняння динаміки, – в якому слід налаштувати значення параметрів конкретного об'єкта. У нейромережевої моделі шаблоном є структура ШНМ, а налаштувати потрібно коефіцієнти міжнейронних зв'язків. Щоб підкреслити особливості використання нейромережевої моделі, вводяться поняття *нейромережевого емулятора і нейромережевого предиктора* [40].

Однокроковий предиктор отримує інформацію про стан об'єкта біля самого об'єкта, що дозволяє робити прогноз стану на один крок уперед (рис. 8.28).

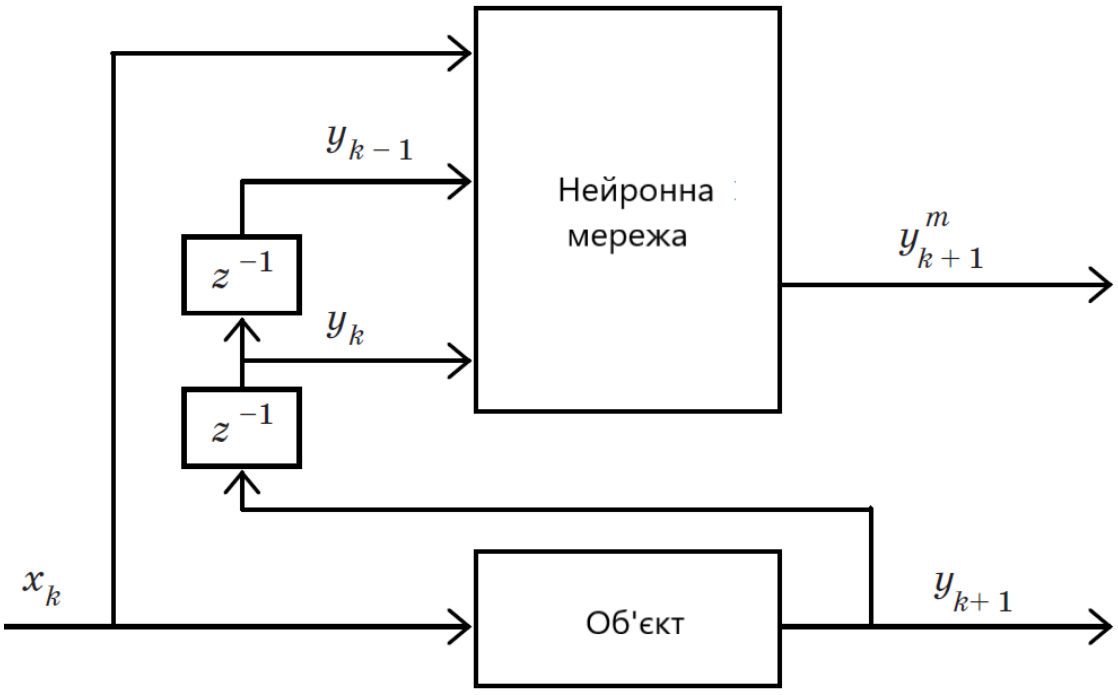


Рисунок 8.28 – Нейромережевий предиктор

Нейромережевий емулятор замість виходу об'єкта використовує вихід нейронної мережі. Це дозволяє робити прогноз динаміки об'єкта на багато кроків уперед, проте помилка тут накопичується зі збільшенням тривалості прогнозу (рис. 8.29).

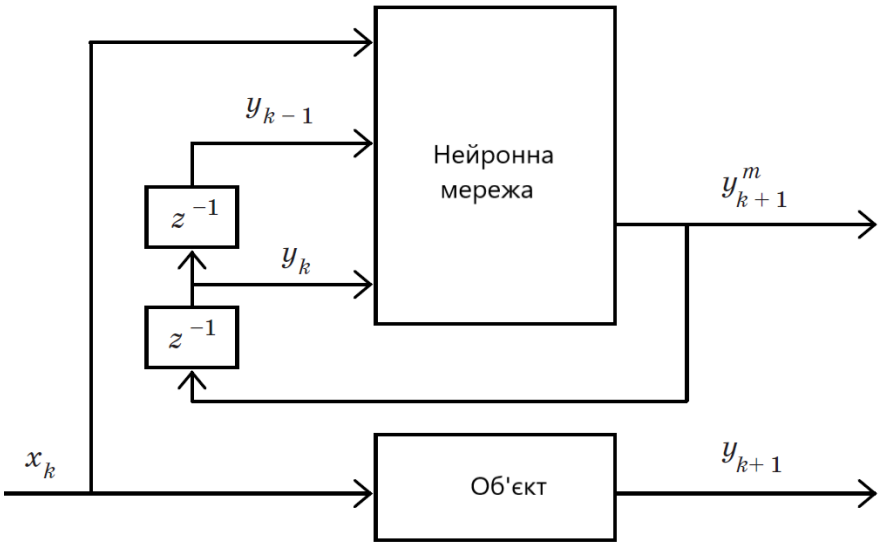


Рисунок 8.29 – Нейромережевий емулятор

Інтуїтивно зрозуміло, що використання короткострокового предиктора природно у ситуації, коли структура об'єкта значно складніша за структуру ШНМ. Наприклад, у завданнях передбачення біржових котирувань або курсів валют нерозумно намагатися отримати прогноз на тривалий термін через безліч факторів, що впливають на поведінку об'єкта прогнозу. Реалізація ж нейроемулятора можлива, якщо структура ШНМ відповідає складності об'єкта.

8.5 Концепція нейроуправління

Людський мозок вирішує найрізноманітніші завдання щодо забезпепечінню життєдіяльності організму. Наприклад, переміщення з однієї точки простору до іншої вимагає фіксації мети руху та визначення помилки положення, виходячи з якої формуються сигнали керування руховою системою. Така дія відповідає традиційній схемі управління зі зворотним зв'язком (рис. 8.30, де $u(t)$ – сигнал управління; $y(t)$ – реальний вихідний сигнал; $g(t)$ – бажаний вихідний сигнал; $e(t)$ – помилка управління). У рамках цієї структури мозок виступає як регулятор, кістково-м'язова система є об'єктом управління, а органи почуттів відіграють роль датчиків.

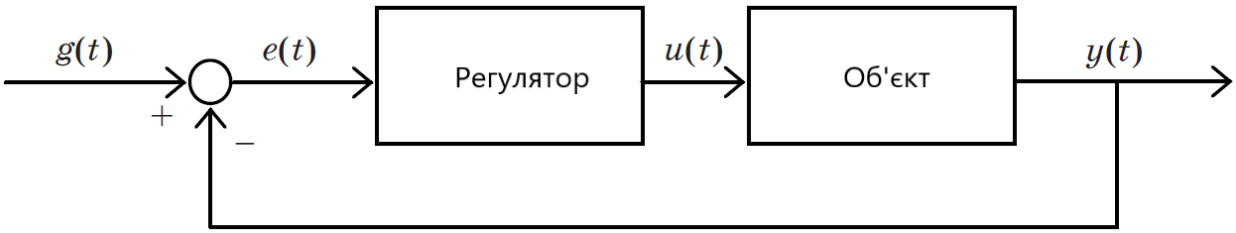


Рисунок 8.30 – Управляння зі зворотним зв'язком

Завдання синтезу регулятора в замкнутій системі управління може розглядатися як завдання синтезу зворотної моделі об'єкта (рис. 8.31, де $g(t)$ – вплив завдання).

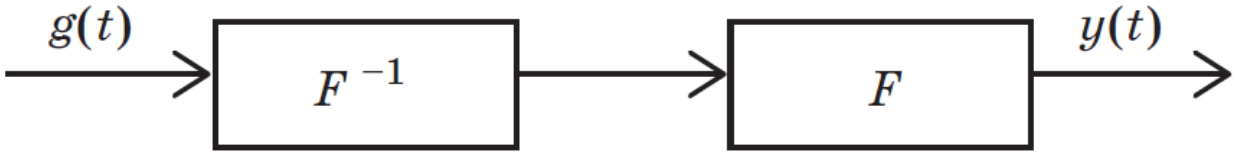


Рисунок 8.31 – Ідеалізована система управління

Якщо F – оператор об'єкта, а F^{-1} – оператор регулятора, то за будь-яких умов виконується рівність

$$y(t) = g(t)\, F^{-1}F = g(t).$$

У реальних умовах точне отримання зворотної моделі об'єкта зазвичай неможливе. Розглянемо проблему на прикладах.

Нехай F - статичний оператор, що реалізує монотонну функціональну залежність $y = F(x)$ (див. рис. 8.32, а). Тоді можна побудувати зворотню залежність $x = F^{-1}(y)$ (див. рис. 8.32,б).

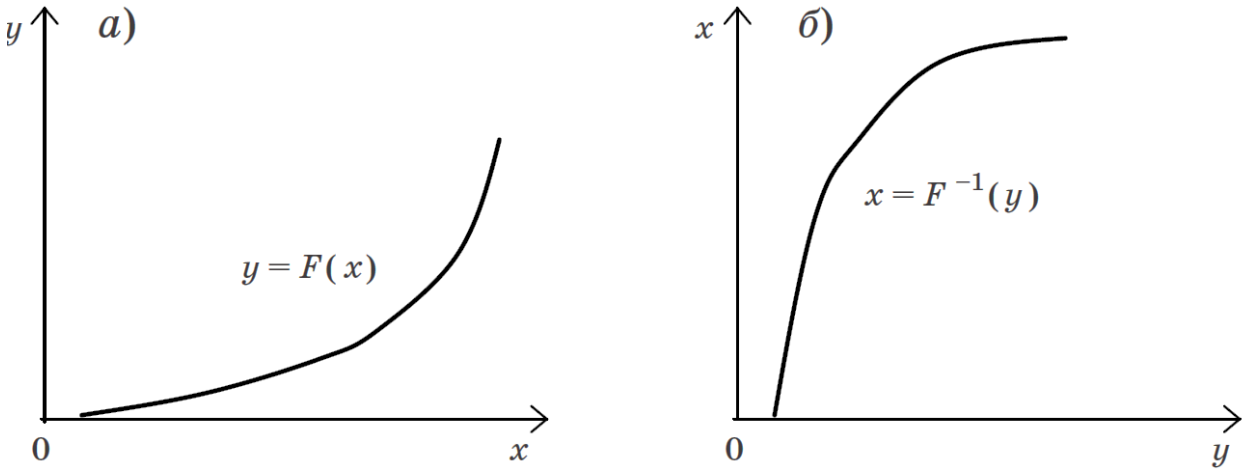


Рисунок 8.32 - Монотонні прямий та зворотний оператор статичної моделі

Якщо оператор моделі не є монотонним, то зворотна залежність не буде функціональною. Отже модель необоротна, як зображено на рис. 8.33.

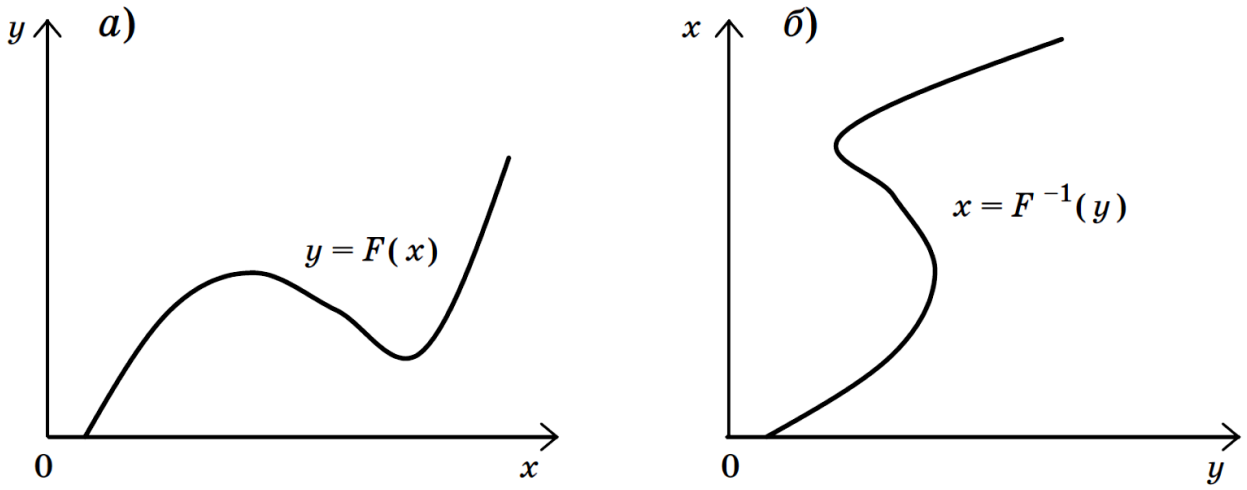


Рисунок 8.33 – Приклад незворотної статичної моделі

Для опису динамічних моделей об'єктів часто використовують апарат передатних функцій (ПФ). Якщо $W(s)$ – передаточна функція об’єкту, тоді для зворотньої моделі $W_{inv}(s)$ повинна виконуватся умова

$$W(s)\,W_{inv}(s)=1.$$

При цьому $W_{inv}(s)$ має бути мінімально-фазовою передатною функцією

Наприклад, пусть $W(s)$ – апереадична ланка 1-го порядку:

$$W(s)=\frac{1}{T_1 s+1}. \qquad (8.1)$$

Тоді

$$W_{inv}(s)=T_1 s+1.$$

Формально умова (8.1) виконується, проте $W_{inv}(s)$ містить операцію ідеального диференціювання, яка фізично нереалізована. Щоб уникнути цього обмеження, замість (8.1) можна розглянути умову

$$W(s)W_{inv}(s)=\frac{1}{T_2 s+1}. \qquad (8.2)$$

Тоді

$$W(s)W_{inv}(s)=\frac{T_1 s+1}{T_2 s+1}.$$

Чим меньш постіяна часу Т2, тем ближче (8.1) та (8.2).

Приклад 8.4. Нехай у задачі керування електромотором математична модель задана передавальною функцією

$$W_{\text{дв}}(s) = \frac{K_{\text{дв}}}{T_{\text{е}}T_{\text{м}}s^2 + (T_{\text{е}} + T_{\text{м}})s + 1}.$$

де $K_{\text{дв}}$ - загальний коефіцієнт підсилення двигуна; $T_{\text{м}}$ - постійна механічна часу, с; $T_{\text{е}}$ - постійна електрична часу, с.

Розглянемо завдання синтезу ПІД-регулятора, передатна функція якого має таку форму

$$W_p(s) = K_p + K_i\frac{1}{s} + K_d s. \qquad (8.3)$$

де K_p, K_i, K_d – невідомі коефіцієнти.

Припустимо, що потрібно забезпечити аперіодичний перехідний процес у замкнутій системі, заданий передавальною функцією

$$W_{\text{ж}}(s) = \frac{1}{T_{\text{ж}}s + 1}.$$

Прирівняємо передавальну функцію бажаної системи до передавальної функції замкнутої системи

$$\frac{1}{T_{\text{ж}}s + 1} = \frac{W_p(s)W_{\text{дв}}(s)}{1 + W_p(s)W_{\text{дв}}(s)}$$

та знайдемо передатну функцію регулятора

$$W_p(s) = \frac{1}{T_{\text{ж}}sW_{\text{дв}}(s)}.$$

Після підстановки в останню формулу значення $W_{\text{дв}}(s)$ отримаємо

$$W_p(s) = \frac{T_{\text{е}} + T_{\text{м}}}{T_{\text{ж}}K_{\text{дв}}} + \frac{1}{T_{\text{ж}}K_{\text{дв}}}\frac{1}{s} + \frac{T_{\text{е}}T_{\text{м}}}{T_{\text{ж}}K_{\text{дв}}}s.$$

У цьому прикладі виявилося можливим отримати аналітичне рішення задачі синтезу регулятора, який є зворотною моделлю об'єкта.

Однак на практиці аналітичне рішення зазвичай неможливе. Застосування нейронних мереж дозволяє знаходити наближені рішення цього завдання.

Передавальної функції ПІД-регулятора (8.3) відповідає формула

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}.$$

Такому регулятору можна поставити у відповідність єдиний нейрон з лінійною активаційною функцією (див. рис. 8.34).

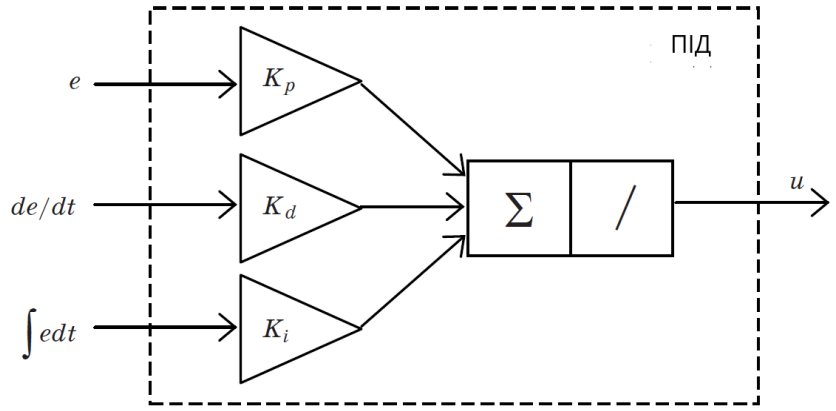


Рисунок 8.34 – ПІД-регулятор як нейроноподібна структура

Штучна нейронна мережа як універсальний налаштовуваний елемент може бути використана в різних постановках завдань при аналізі і синтезі систем автоматичного управління.

У системах управління найчастіше застосовується ШНМ ПП, яка може виступати як ідентифікаційна модель об'єкта управління або регулятора [26].

В теорії автоматичного управління динамічні ланки (об'єкти управління і регулятори) часто описуються передатною функцією ([41] та інших.):

$$W(s) = \frac{y(s)}{x(s)} = \frac{s^m b_m + s^{m-1} b_{m-1} + \cdots + s b_1 + b_0}{s^n a_n + s^{n-1} a_{n-1} + \cdots + s a_1 + a_0},$$

де y - вихідний сигнал ланки; x – вхідний сигнал; a_n та b_m – постійні коефіцієнти; $y(s)$ та $x(s)$ – зображення по Лапласу виходу та входу ланки, $m \le n$.

Цей вираз можна записати у вигляді

$$y(s)(s^n a_n + s^{n-1} a_{n-1} + \cdots + s a_1 + a_0) = \\ = x(s)(s^m b_m + s^{m-1} b_{m-1} + \cdots + s b_1 + b_0).$$

При цифровому моделюванні розглядається дискретне уявлення передавальної функції. Процес дискретизації ПФ полягає у заміні операторів диференціювання відносинами кінцевих різниць

$$sx = \frac{x_k - x_{k-1}}{\Delta t}, \quad s^2x = \frac{x_k - 2x_{k-1} + x_{k-2}}{\Delta t^2}, \ldots$$

де Δt – крок дискретизації за часом; k – номер моменту часу.

Після всіх перетворень та спрощень отримуємо формулу виду

$$y_n = \sum_{i=1}^{n} \acute{a}_i y_{n-i} + \sum_{j=1}^{m} \acute{b}_i y_{m-j},$$

де $\acute{a}_i$ та $\acute{b}_i$ – постійні коефіцієнти, яки залежать від кроку дискритизації за часом.

Цього ж результату можна досягти з використанням z-перетворення, при якому виходить дискретна ПФ, що має наступний стандартний вигляд

$$W(z) = \frac{y(z)}{x(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{a_0 + a_1 z^{-1} + \cdots + a_m z^{-m}},$$

де оператор z^{-1} означає затримку і тактів.

Слід відмітити, що з безперервної ПФ можна отримати безліч варіантів дискретної ПФ при різних періодах дискретизації. Дискретній ПФ відповідає структура, показана на рисунку 8.35. Часто її називають цифровим фільтром.

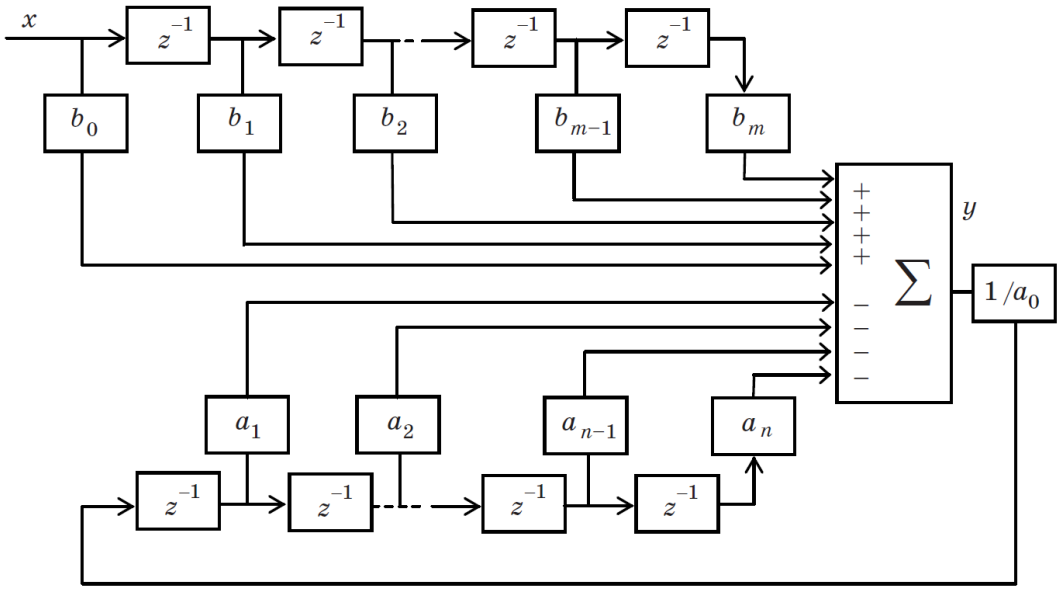


Рисунок 8.35 – Структура цифрового фільтра

У реальних системах кількість ліній затримки зазвичай невелика. Наприклад, при цифровій реалізації ПІД-регулятора можна використовувати формулу виду

$$y = z^{-1}y + b_1x + b_2z^{-1}x + b_3z^{-2}x.$$

Структура на рисунке 8.36 очевидно нагадує нейрон із зворотними зв'язками та лінійною активаційною функцією. Обмежені можливості такої системи були проаналізовані у попередніх розділах

Використання багатошарових ШНМ ПП дає більш широкі можливості моделювання, оскільки їхня поведінка може бути нелінійною.

Існує два підходи до застосування нейронних регуляторів (або нейроконтролерів (НК)) ([42 – 45] та інш.):

- *прямі методи*, засновані на безпосередньому управлінні об'єктом за допомогою нейрорегуляторів (НР);

– *непрямі методи*, коли ШНМ використовується для виконання допоміжних функцій управління, таких як фільтрація шуму або ідентифікація динамічного об'єкта.

Залежно від числа нейронних мереж нейроконтролери можуть бути *одномодульними або багатомодульними*.

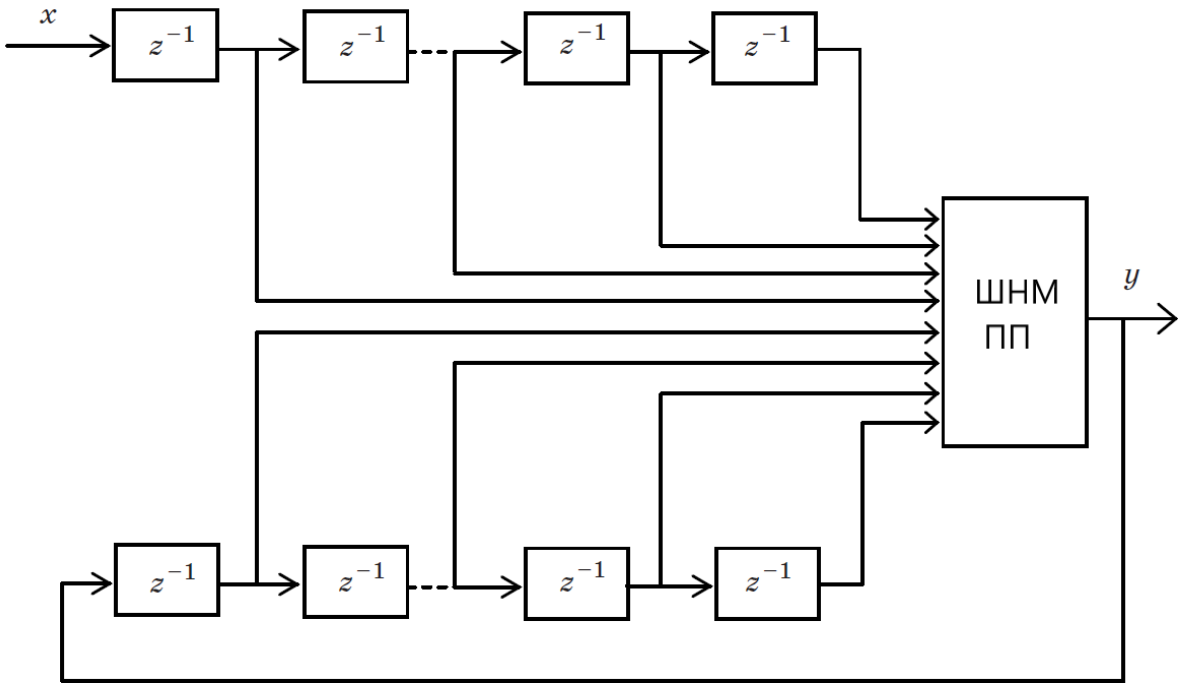


Рисунок 8.36 - Загальна стурктура нейронного регулятору

Схеми нейроуправління, у яких нейроконтролери застосовуються разом із традиційними контролерами, називаються *гібридними*.

У задачах нейроуправління для представлення об'єкта управління використовують модель типу «чорний ящик», в якому спостерігаються поточні значення входу і виходу. Стан об'єкта вважається недоступним для зовнішнього спостереження, хоча розмір вектора станів зазвичай приймається фіксованою. Динаміку поведінки об'єкта управління можна у дискретному вигляді:

$$\begin{cases} X(k+1) = F\big(X(k), U(k)\big), \\ Y(k+1) = G\big(X(k)\big), \end{cases}$$

де $X(k) \in R^n$ - значення вектора стану на такті k; $U(k) \in R^m$ – значення вектора входу (управління); $Y(k+1) \in R^p$ – значення вектора виходу на такті k+1.

Для оцінки вектора стану динамічного об'єкта порядку n може бути використана *модель нелінійної авторегресії з додатковими вхідними сигналами (NARX)*, яка для одномірного об'єкта набуває вигляду

$$X(k) = [y(k), y(k-1), \cdots, y(k-n), u(k), u(k-1), \cdots, u(k-m)]^T.$$

Цю модель часто спрощують, приводячи до моделі виду

$$X(k) = [y(k), y(k-1), \cdots, y(k-n)]^T.$$

Таким чином, на підставі цього виразу та рис. 8.36 структуру системи управління можна зобразити так, як показано на рис. 8.37 (Δ – позначення лінії затримки).

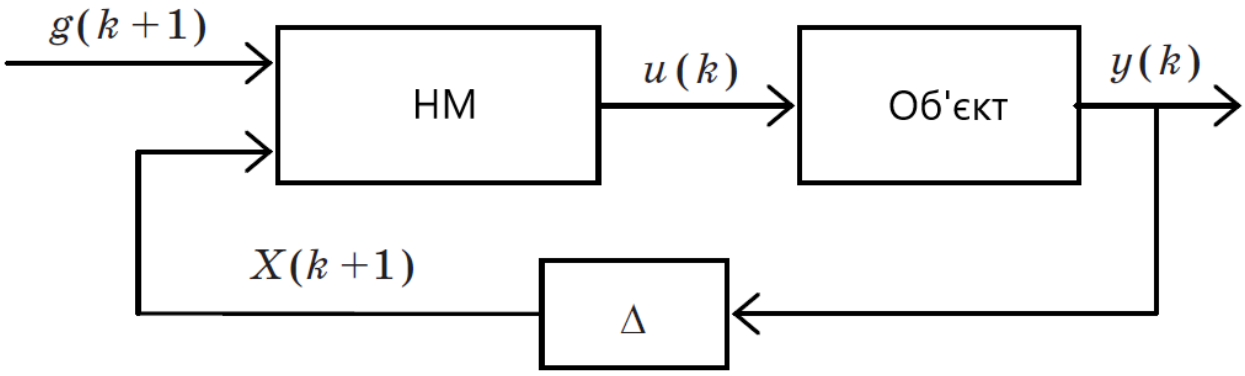


Рисунок 8.37 - Нейроуправління за станом

Будемо вважати, що в схемах ШНМ, що наводяться далі, на вході маються необхідні лінії затримки.

8.6 Інверсне нейрокерування

При інверсному нейроуправлінні формування інверсної моделі об'єкта управління здійснюється шляхом навчання нейронної мережі. Відомо кілька видів такого нейроуправління, орієнтованих використання при навчанні алгоритму зворотного поширення помилки [26, 42, 43].

У схемі *прямого інверсного управління* (Direct Inverse Neurocontrol або Generalized Inverse Neurocontrol) ШНМ розглядається як інверсна модель динаміки системи (рис. 8.38).

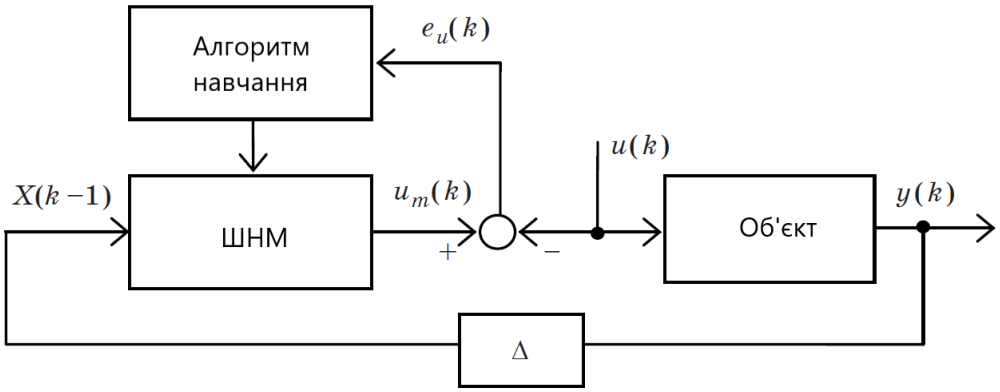


Рисунок 8.38 - Пряме інверсне нейроуправління

Навчання ШНМ відбувається в режимі offline, причому сигнал *u*(*t*) тут виступає як тестовий, тому він може бути деяким випадковим процесом.

У ході навчання ШНМ повинна апроксимувати залежність значень сигналу *u*(*k*), що управляє, від попереднього стану *X*(*k* − 1). Таким чином, навчальну вибірку формує безліч пар виду

$$\{X(k-1), u(k)\}_i, i, \qquad i = 1, 2. \cdots, N.$$

Основна труднощі використання алгоритму зворотного поширення помилки у межах цієї схеми у тому, що необхідно перебувати на околиці глобального мінімуму помилки, але це вимагає досить точного знання структури системи. Тому ефективнішим може бути використання алгоритмів глобальної оптимізації, таких, як *генетичний алгоритм*.

Після навчання ШНМ використовується безпосереднього управління об'єктом.

Перевагою прямого інверсного нейроуправління є можливість навчання у режимі offline і відсутність необхідності у точної математичної моделі об'єкта управління. До недоліків методу слід віднести складність формування навчальної вибірки через необхідність ретельного підбору випадкового ідентифікуючого процесу, що подається на вхід системи, а також низька якість роботи в тих випадках, коли інверсія об'єкта управління виявляється неоднозначною функцією. Неоднозначність призводить до появи протиріч у навчальній вибірці, що заводять у тупик процес навчання нейронної мережі.

Для вирішення цієї проблеми пропонується низка методів.

У схемі *непрямого інверсного керування*, наведеної на рис. 8.39 використовуються дві копії ШНМ як інверсної моделі [26].

Вхідний сигнал надходить на вхід першої копії ШНМ, яка виробляє сигнал керування. Друга копія ШНМ використовується аналогічно до схеми прямого інверсного навчання.

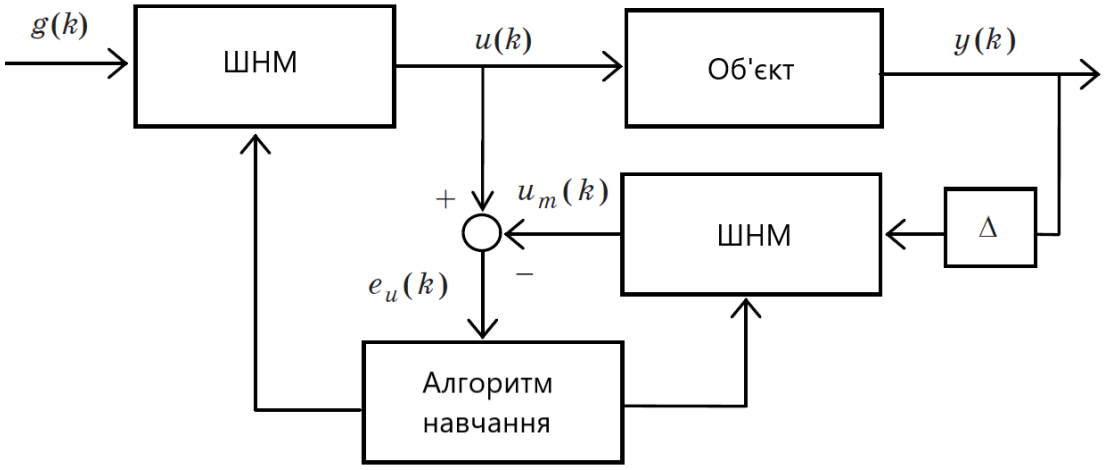


Рисунок 8.39 - Непряме інверсне керування

У випадку регулятор повинен компенсувати відхилення реального виходу об'єкта управління (ОУ) *y*(*t*) від бажаного сигналу *g*(*t*):

$$e_y(t) = y(t) - g(t).$$

Проблема використання схеми непрямого управління полягає в тому, що вона іноді не працює, тому що в процесі навчання ШНМ може відображати велику кількість різних вхідних сигналів на те саме значення управління (вироджене відображення), так що

$$e_u(t) = 0 \text{ при } e_y(t) \neq 0.$$

Тому під час навчання доцільно використовувати помилку виходу об'єкта $e_y(t)$, а чи не помилку управління $e_u(t)$.

У методі *спеціалізованого інверсного нейроуправління* (див. рис. 8.40) проблема навчання інверсній динаміці вирішується шляхом апроксимації аналітичної моделі керованого об'єкта та обчислення локальних значень якобіана для різних галузей простору станів.

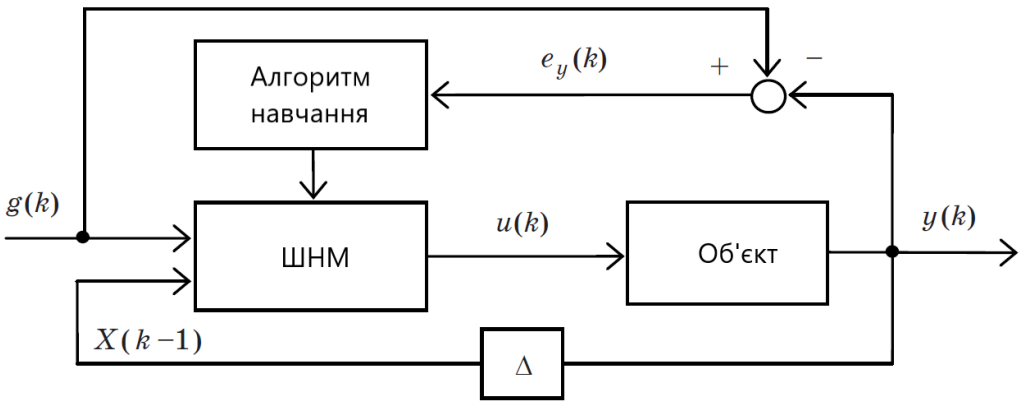


Рисунок 8.40 - Спеціалізоване інверсне керування

Корекція вагових коефіцієнтів нейронної мережі виконується методом якнайшвидшого спуску:

$$w(k+1) = w(k) - \Delta w(k),$$

$$\Delta w(k) = -\alpha e(k) \frac{\partial y(k+1)}{\partial u(k)} \frac{\partial u(k)}{\partial w(k)},$$

де α - коефіцієнт швидкості навчання.

Величина похідної $\frac{\partial u(k)}{\partial w(k)}$ обчислюється методом зворотного поширення помилки, а похідна $\frac{\partial y(k+1)}{\partial u(k)}$ є якобіан об'єкта управління, значення якого можна знайти аналітично за заданою математичною моделлю об'єкта управління. Насправді щоб одержати прийнятного якості управління часто буває достатньо обчислити лише знак якобіана [26]. Корекція значень вагових коефіцієнтів продовжується до досягнення прийнятної якості керування.

Нейронна мережа навчається з метою мінімізації помилки керування $e_y(t)$. Метод може забезпечити оптимальне рішення, якщо помилка надійно вимірюється.

Проблема спеціалізованого інверсного навчання при використанні алгоритму зворотнього розповсюдження помилки (АЗРП) полягає в тому, що в цій схемі $e_y(t)$ виявляється нескорельованою з помилкою навчання інверсного контролера eu(t), тобто вона може мати інші знак та амплітуду.

У той самий час будь-яка досить груба апроксимація $e_u(t)$ з допомогою $e_y(t)$ дозволяє процесу навчання сходитися.

Щоб уникнути проблем спеціалізованого інверсного навчання було розроблено *схему зворотного поширення помилки у часі* [26, 43].

Цей алгоритм ґрунтується на ідеї застосування двох нейронних мереж, одна з яких виконує функцію нейроконтролера, а друга – функцію нейроемулятора, який навчається моделювати динаміку об'єкта управління.

У першому етапі відбувається навчання нейроемулятора як offline. Нейроемулятор вважається навченим, якщо за однакових значеннях з його входах і входах реального об'єкта відмінність їх значень на виходах стає незначним. Після закінчення навчання нейроемулятор проводиться навчання нейроконтролера. Навчання виконується в режимі on line за такою ж схемою, як і у разі інверсного спеціалізованого нейроуправління (див. рис. 8.41).

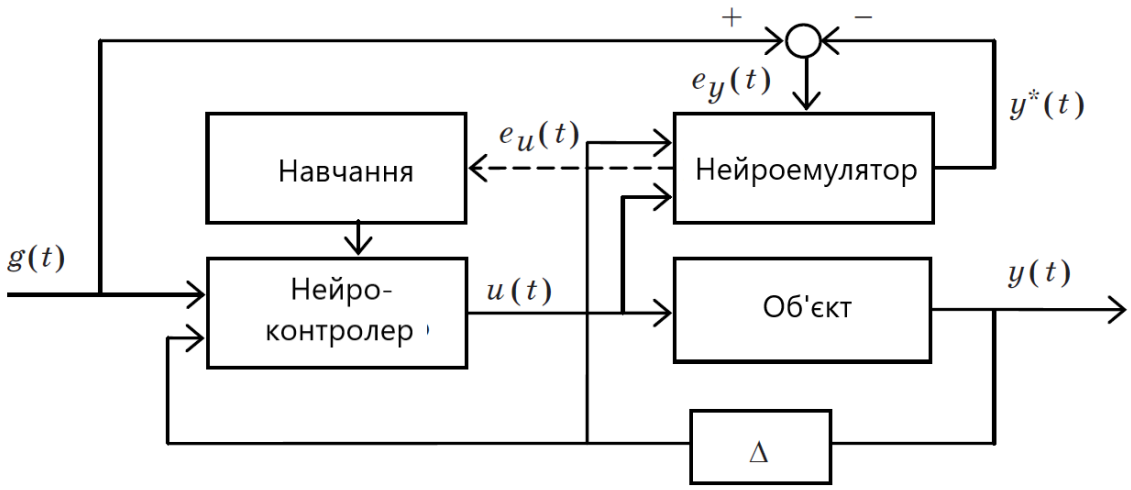


Рисунок 8.41 - Зворотне поширення у часі

У процесі навчання НК поточна помилка управління пропускається через нейроемулятор у зворотному напрямку.

Алгоритм дозволяє емулювати $e_u(t)$ за допомогою використання помилки зворотного розповсюдження $e_y(t)$ через модель передбачення (forward model). Тут помилка $e_y(t)$, поширюючись до вхідного шару, буде відповідати помилці $e_u(t)$. Це робить схему навчання придатною для використання у більшості лінійних систем.

Механізм зворотного проходження помилки через нейроемулятор реалізує локальну інверсну модель у точці простору станів об'єкта управління. Пройшовши через нейроемулятор, помилка поширюється далі через нейроконтролер, але її проходження супроводжується корекцією вагових коефіцієнтів НК. Нейроемулятор виконує функції додаткових шарів нейронної мережі НК, у яких ваги зв'язків не коригуються.

8.7 Нейроконтролери в MatLab

Нейронні мережі успішно застосовуються під час синтезу систем управління динамічними процесами. Універсальні можливості апроксимації багатошарових ШНМ прямого поширення дозволяють вирішувати завдання ідентифікації, проектування та моделювання нелінійних систем управління.

У пакеті прикладних програм Neural Network toolbox реалізовано три варіанти контролерів:

1. Контролер із передбаченням (NN Predictive Controller).
2. Контролер з урахуванням авторегресії зі ковзним середнім (NARMA – L2 Controller).
3. Контролер з урахуванням еталонної моделі (Model Reference Controller).

Всі ці варіанти вимагають виконання двох етапів проектування:

ідентифікації керованого процесу та синтезу закону управління.

На етапі ідентифікації розробляється модель керованого процесу у вигляді ШНМ, яка потім використовується для синтезу регулятора.Схема ідентифікації керованого процесу наведено на рис. 8.42. Вона включає модель управління у вигляді ШНМ, яка повинна бути навчена в автономному режимі так, щоб мінімізувати помилку між виходами об'єкта і моделі: $e = y_p - y_m$ для послідовності пробних вхідних сигналів u.

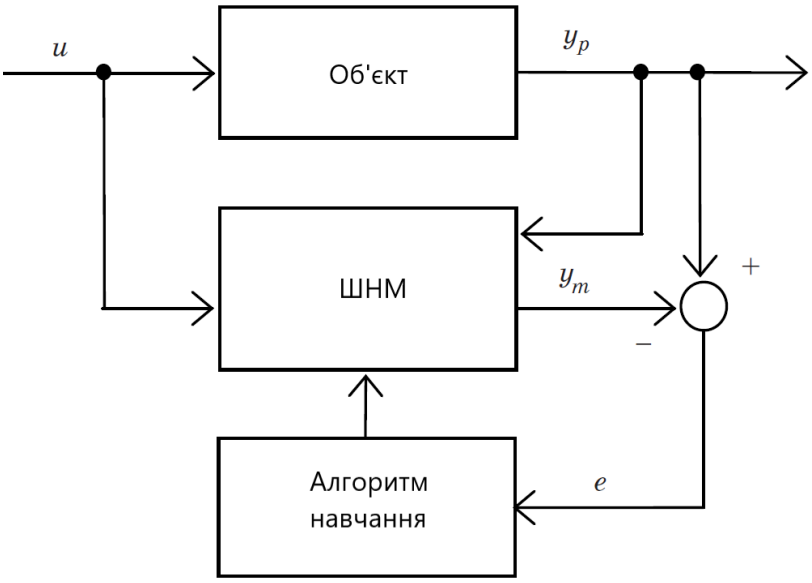


Рисунок 8.42 - Загальна схема нейромережевої ідентифікації у MatLab

Для ідентифікації застосовується двошарова ШНМ із лініями затримки (див. рис. 8.43). Модель використовує попередній вхід та попередній вихід об'єкта для передбачення його майбутнього виходу.

Модель навчається в режимі offline, використовуючи накопичену раніше інформацію про поведінку об'єкта.

Налаштування ШНМ здійснюється за даними випробувань реального об'єкта. Метод навчання може бути будь-яким із наявного набору.

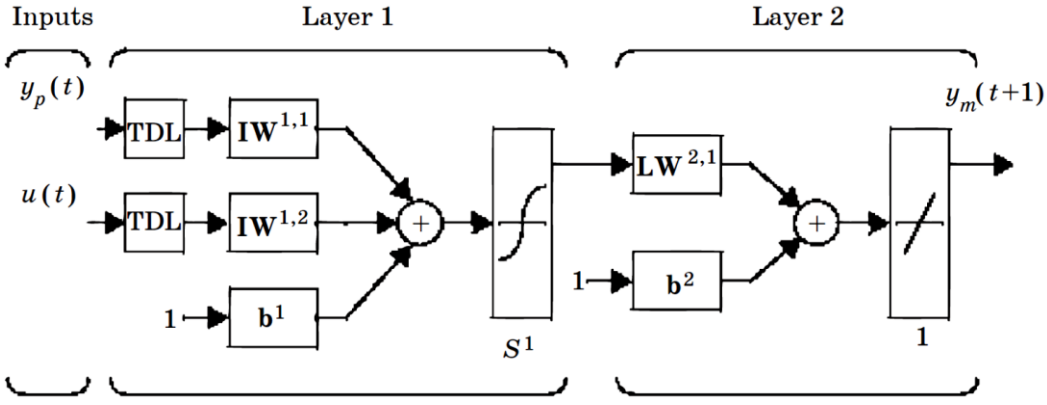


Рисунок 8.43 – Ідентифікаційна нейрона мережа

Хоча для кожної з трьох архітектур використовується та сама процедура ідентифікації, етапи синтезу істотно різні.

При *управлінні з передбаченням* модель керованого процесу у вигляді ШНМ використовується для передбачення його майбутньої поведінки, а алгоритм оптимізації застосовується для розрахунку такого управління, яке мінімізує різницю між бажаним та дійсними змінами виходу моделі. Контролер, реалізує такий регулятор, вимагає значного обсягу обчислень, оскільки розрахунку оптимального закону управління оптимізація виконується кожному такті управління.

Регулятор із передбаченням використовує модель нелінійного керованого процесу у вигляді ШНМ для передбачення його майбутньої поведінки. Крім того, регулятор обчислює сигнал керування для оптимізації поведінки об'єкта на заданому інтервалі. часу.

Управління з передбаченням будується на принципі горизонту, що віддаляється, згідно з яким нейромережева модель керованого процесу передбачає реакцію об'єкта управління на певному інтервалі часу в майбутньому. Передбачення використовується програмою оптимізації для обчислення керуючого сигналу, який мінімізує критерій якості управління [46]:

$$J = \sum_{j=N_1}^{N_2} \left(y_r(t+j) - y_m(t+j)\right)^2 + \rho \sum_{j=1}^{N_u} \left(\acute{u}(t+j-1) - \acute{u}(t+j-2)\right)^2,$$

де константи N_1, N_2 і N_u задають межі, всередині яких обчислюються помилка стеження та потужність керуючого сигналу. Змінна u' описує пробний керуючий сигнал, y_r та y_m – бажана та реальна реакції моделі. Розмір ρ визначає внесок потужності управління.

Структурна схема, наведена на рис. 8.44 ілюструє процес управління з передбаченням.

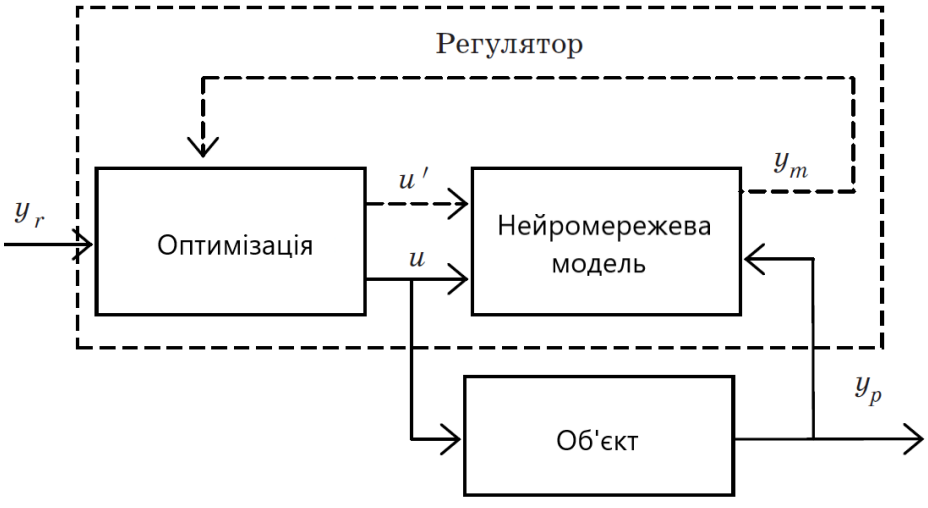


Рисунок 8.44 - Структурна схема регулятора із передбаченням

Регулятор складається з нейромережевої моделі та блоку оптимізації. Блок оптимізації обчислює значення *u'*, які мінімізують критерій якості керування, а відповідний сигнал керує процесом.

У MATLAB нейрорегулятор з передбаченням задається блоком NN Predictive Controller, він знаходиться у бібліотеці інструментів MatLab у папці Neural Network Blockset/Control Systems.

Приклад 8.4. Нейрорегулятор із передбаченням (NN Predictive Controller)

Розглянемо приклад використання нейрорегулятора з передбаченням, описаний Neural Net toolbox, для об'єкта управління, показаного на рис. 8.45.

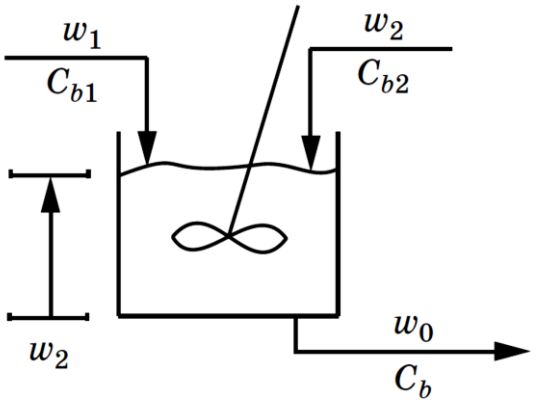


Рисунок 8.45 - Каталітичний реактор з безперервним перемішуванням

Динамічна модель системи описується рівняннями

$$\begin{cases} \dfrac{dh(t)}{dt} = w_1(t) + w_2(t) - 0{,}2\sqrt{h(t)}, \\ \dfrac{dC_b(t)}{dt} = (C_{b1} - C_b(t))\dfrac{w_1(t)}{h(t)} + (C_{b2} - C_b(t))\dfrac{w_2(t)}{h(t)} - \dfrac{k_1 C_b(t)}{\left(1 + k_2 C_b(t)\right)^2}, \end{cases}$$

де $h(t)$ - рівень рідини; $C_b(t)$ - концентрація продукту на виході; w_1 та w_2 – швидкість потоків компонентів суміші; C_{b1} та C_{b2} – концентрації компонентів суміші: C_{b1} = 24,9, C_{b2} = 0,1; k_1 та k_2 – константи процесу k_1 =k_2 =1.

З метою спрощення моделі прийнято w_1 = const = 0,1, і сигналом управління є величина w_2.

Приклад викликається командою predcstr (див. рис. 8.46).

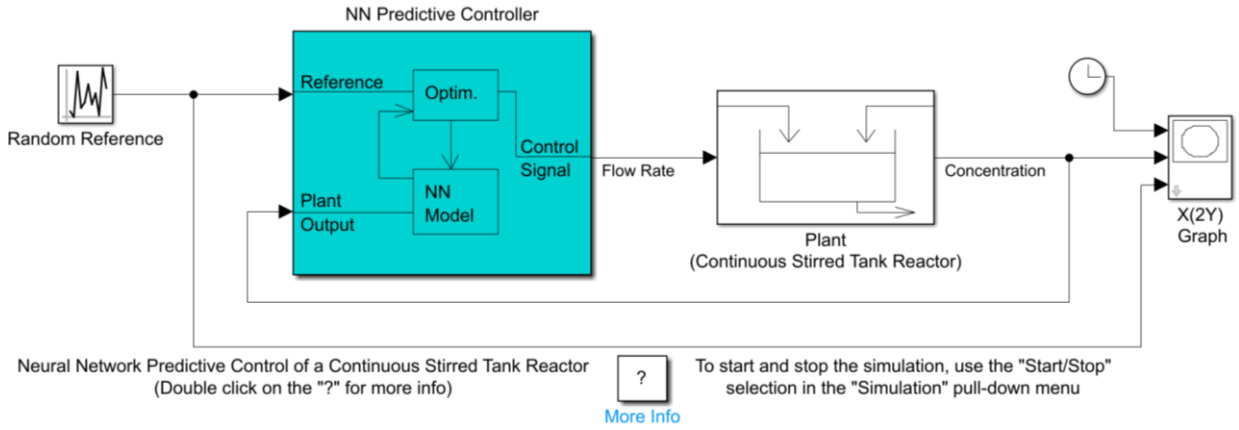


Рисунок 8.46 - Математична модель системи керування змішувальним баком

При натисканні на блок NN Predictive Controller відкривається вікно, наведене на рис. 8.47.

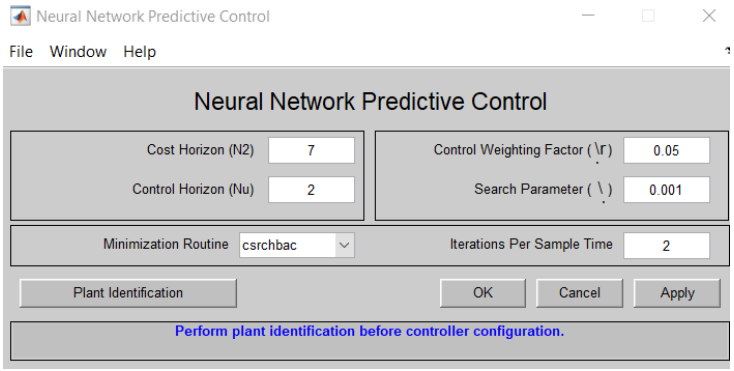


Рисунок 8.47 - Інтерфейс регулятора із передбаченням

Регулятор має такі параметри:

- Cost Horizon (N2) – верхня межа підсумовування у показнику якості N2 = 7, нижня межа фіксована: N2 = 1;
- Control Horizon (Nu) – верхня межа підсумовування в оцінці потужності управління;
- Control Weighting Factor (ρ) – коефіцієнт ваги для складової потужності керування ρ = 0,05;
- Search parameter (α) – параметр одновимірного пошуку, що задає поріг зменшення якості α = 0,001;
- Minimization Routine – вибір процедури одновимірного пошуку;

• Iteration Per Sample Time – кількість ітерацій на один такт дискретності.

Напис у нижній частині вікна наказує виконати ідентифікацію до встановлення параметрів регулятора. Панель ідентифікації наведено на рисунку 8.48.

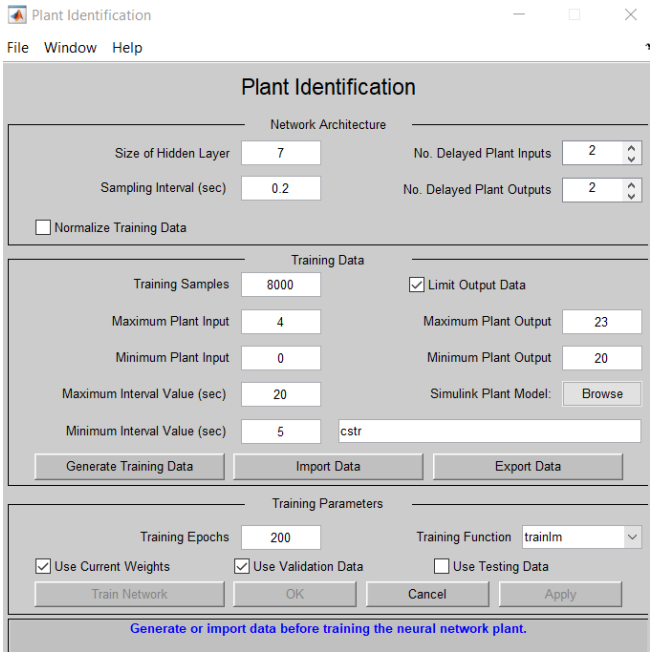


Рисунок 8.48 - Інтерфейс ідентифікації об'єкта

Набір елементів, що управляють, для завдання архітектурних параметрів нейронної мережі наступний:

- Size of the Hiden Layer – кількість нейронів у прихованому шарі;
- No. Delayed Plant Inputs – кількість затримок для вхідного шару;
- No. Delayed Plant Outputs – кількість затримок для вихідного шару;
- Samling Interval – інтервал квантування або крок дискретності за секунди між двома послідовними моментами відліку даних;
- Notmalize Training Data – перемикач нормування для перетворення навчальних даних до діапазону [0 1].

Набір керуючих елементів для завдання характеристик навчальної послідовності включає в себе:

- Training Samples – кількість точок відліку для отримання навчальної послідовності у вигляді пар значень вхід-вихід для керованого процесу, що визначається моделлю Simulink;
- Maximum Plant Input – максимальне значення вхідного сигналу;
- Minimum Plant Input – мінімальне значення вхідного сигналу;
- Maximum Interval Value (sec) – максимальний інтервал ідентифікації (у секундах);
- Minimum Interval Value (sec) – мінімальний інтервал ідентифікації (у секундах);
- Limit Output Data – перемикач обмеження значень вихідного сигналу;
- Maximum Plant Output – максимальне значення вихідного сигналу, яке задається при включеному перемикачі Limit Output Data;
- Minimum Plant Output – максимальне значення вихідного сигналу, яке задається при включеному перемикачі Limit Output Data;

- Simulink Plant Model – вікно завдання моделі керованого процесу, реалізованої за допомогою блоків Simulink, що має порти входу та виходу та збереженої у файлі *.slx; вибір моделі провадиться за допомогою кнопки Browse; Ім'я моделі відображається у спеціальному вікні.

Параметри навчання задаються таким чином:

- Training Epochs – кількість циклів навчання;
- Training Function – для завдання навчальної функції;
- Use Current Weights – перемикач для використання поточних ваг нейронної мережі;
- Use Validation Data – перемикач для використання контрольної множини в обсязі 25% від навчальної множини;
- Use Testing Data – перемикач для використання тестової множини в обсязі 25% від навчальної множини.

Для ідентифікації керованого процесу необхідно виконати такі дії:

- встановити архітектуру ШНМ, яка буде моделлю керованого процесу;
- встановити параметри навчання;
- вибрати модель Simulink для керованого процесу;
- згенерувати навчальну послідовність заданого обсягу, запустивши модель Simulink за допомогою кнопки Generate Training Data.

Генерація навчальної послідовності проводиться за допомогою впливу ряду ступінчастих сигналів на математичну модель процесу управління та зняття значень на вході та виході моделі через кожен крок квантування. Графіки вхідного та вихідного сигналів відображаються у вікні Plant Input-Output Data (див. рис. 8.49).

Після завершення генерації навчальної послідовності необхідно або прийняти ці дані, натиснувши кнопку Accept Data, і тоді вони будуть використані для навчання нейронної мережі, або відкинути їх, натиснувши кнопку Reject Data.

Після отримання навчальної послідовності необхідно встановити необхідні параметри навчання та за допомогою кнопки Train Network (див. рис. 8.48) запустити процес навчання нейронної мережі. Після закінчення навчання його результати відображаються на графіках зміни помилки мережі для навчальної та тестової послідовності, а також у вікні вихідних значень моделі та мережі при подачі на вхід зазначених послідовностей.

Результати навчання НМ наведені на рисунку 8.50.

Якщо результати навчання прийнятні, необхідно зберегти параметри нейромережевой моделі керованого процесу і розпочати синтезу регулятора, натиснувши кнопки Apply і OK. В іншому випадку слід натиснути кнопку Cancel і повторити процес ідентифікації, спочатку змінивши архітектуру мережі та параметри навчальної послідовності.

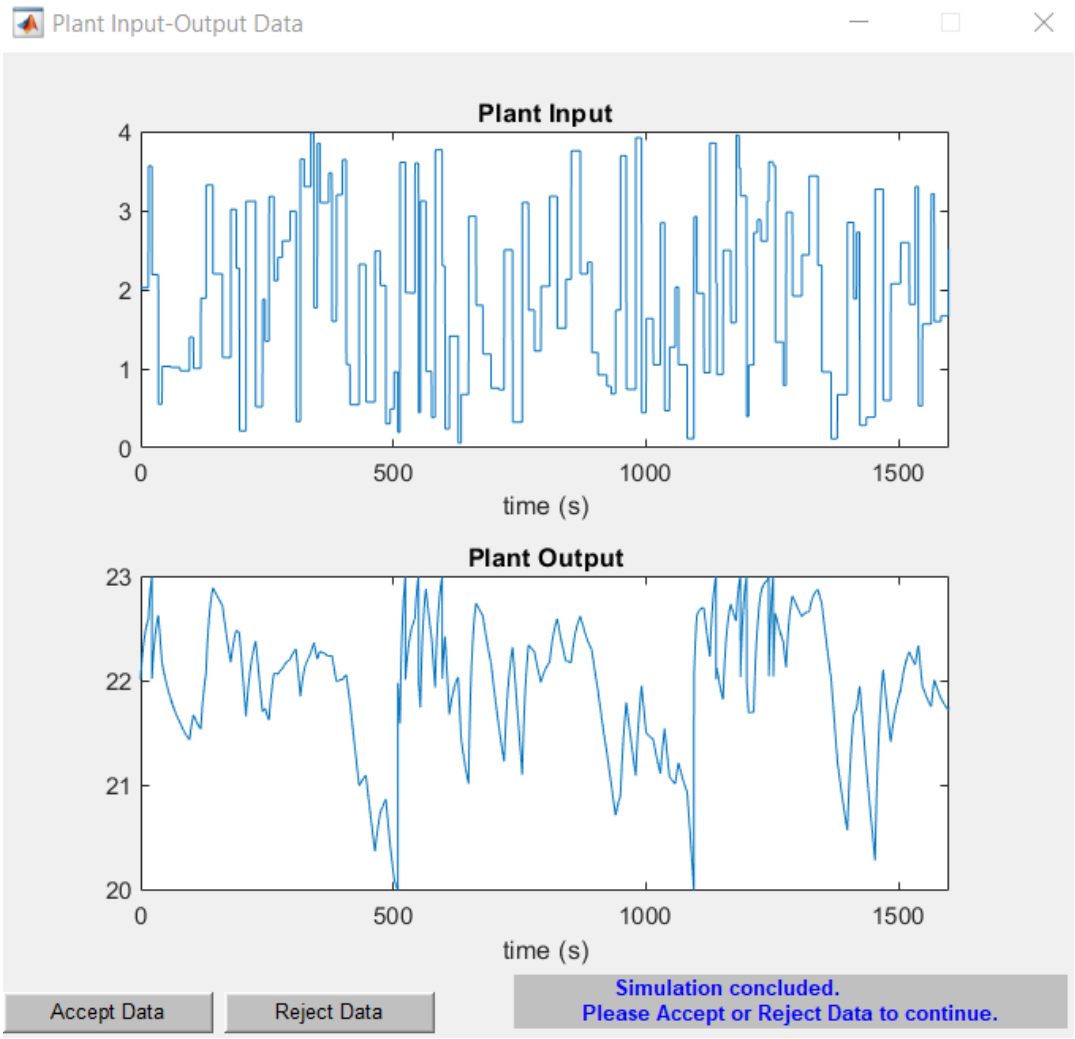


Рисунок 8.49 – Дані для навчання нейронної мережі

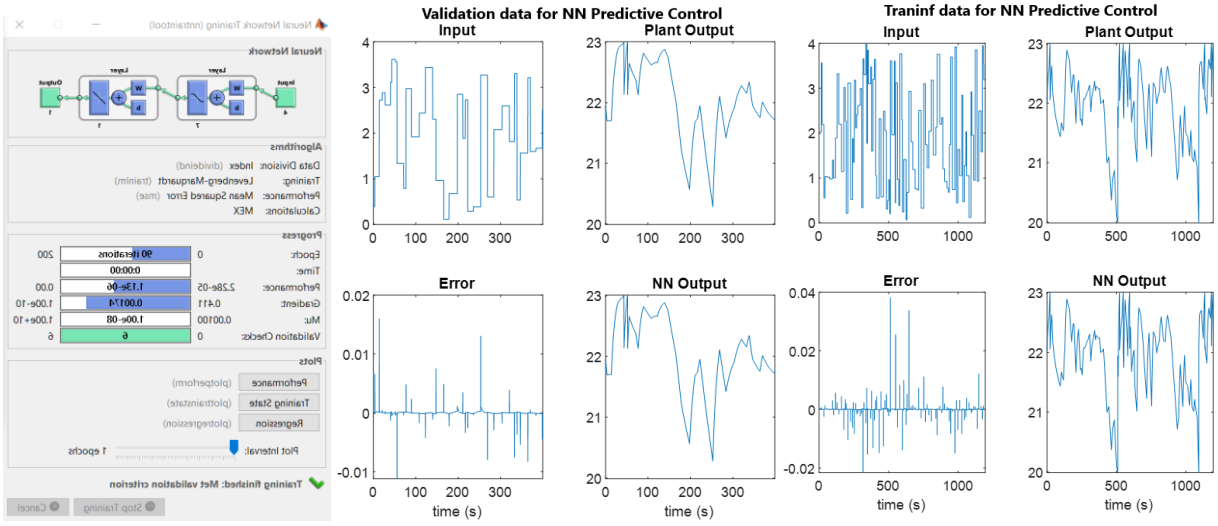


Рисунок 8.50 – Результати навчання нейронної мережі

Навчальну послідовність можна імпортувати з робочої області або файлу, натиснувши кнопку Import Data. Кнопка Export Data дозволяє зберегти навчальну послідовність у робочій області або у файлі.

Кнопка Erase Generated Data видаляє згенеровані дані.

Таким чином, діалогова панель Plant Identification дозволяє ідентифікувати керований процес, представлений у вигляді моделі Simulink, побудувати двошарову нейронну мережу прямого поширення з необхідним числом нейронів та ліній затримки, і навчити цю мережу для отримання нейронної моделі керованого процесу, а також оцінити якість навчання та роботу нейронної мережі.

Після завершення процесу ідентифікації необхідно повернутися до вікна параметрів регулятора, натиснути кнопки Apply та OK, після чого можна розпочати безпосередньо моделювання.

На рис. 8.51 наведено приклад відпрацювання системою ступеневих вхідних сигналів з видатковим періодом та амплітудою.

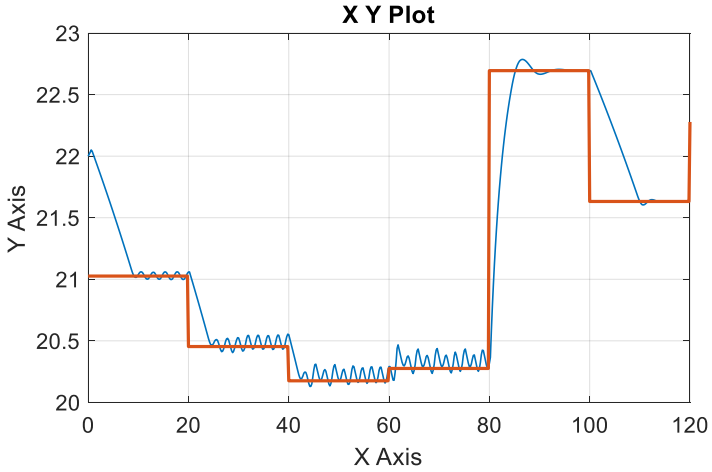


Рисунок 8.51 – Результати функціонування регулятора з передбаченням

При управлінні на основі *авторегресії зі ковзним середнім* регулятор є досить простою реконструкцією моделі керованого процесу. Такий регулятор потребує найменшого обсягу обчислень. Обчислення у реальному часі пов'язані лише з реалізацією нейронної мережі. Недоліком методу є те, що модель процесу повинна бути задана в канонічній формі простору стану, що може призводити до обчислювальних похибок.

Принципова ідея NARMA-L2 (Nonlinear Autoregressive - Moving Average (нелінійна авторегресія зі ковзним середнім)) полягає у поданні нелінійної системи у вигляді лінійної моделі [47].

Модель NARMA має вигляд

$$y(k+d) = N[y(k), y(k-1), \cdots, y(k-n+1), u(k), u(k-1), \cdots, u(k-n+1)]$$

де $y(k)$ – вихід моделі; d - число тактів передбачення; $u(k)$ – вхід

моделі.

На стадії ідентифікації необхідно навчити ШНМ апроксимувати нелінійну функцію *N*(*y*,*u*). Ця ідентифікаційна процедура використовує NN Predictive Controller.

Якщо потрібно забезпечити відповідність виходу системи еталонному виходу $y(k+d) = y_r(k+d)$, то наступний крок полягає у розробці нелінійного контролера у формі

$$u(k) = G[y(k), y(k-1), \cdots, y(k-n+1), y_r(k+d), u(k-1), \cdots, u(k-m+1)]$$

Для того, щоб ШНМ апроксимувала функцію *G* з мінімальною середньоквадратичною помилкою, необхідно використовувати динамічний алгоритм її зворотного розповсюдження, що призводить до необхідності надмірного об'єму обчислень.

Рішення, запропоноване в [48], передбачає використання наближеної моделі з виділеною складовою управління. Така модель, що називається NARMA-L2, має вигляд

$$\hat{y}(k+d) = f[y(k), y(k-1), \cdots, y(k-n+1), u(k-1), \cdots, u(k-m+1)] + \\ + g[y(k), y(k-1), \cdots, y(k-n+1), u(k-1), \cdots, u(k-m+1)]u(k).$$

Перевага цієї форми полягає в тому, що тепер поточне управління можна обчислити безпосередньо, якщо відомі бажана траєкторія y_r, передісторія управління $\{u(k-1), \ldots, u(k-m+1)\}$, і навіть попередні і поточне значення виходу $\{y(k), \ldots, y(k-n+1)\}$:

$$u(k) = \frac{y_r(k+d) - f[y(k), y(k-1), \cdots, y(k-n+1), u(k-1), \cdots, u(k-m+1)]}{g[y(k), y(k-1), \cdots, y(k-n+1), u(k-1), \cdots, u(k-m+1)]}.$$

У цьому рівнянні *u*(*k*) залежить від поточного значення *y*(*k*), тому безпосереднє використання його важко. Проте рівняння можна модифікувати так:

$$u(k+1) = \\ = \frac{y_r(k+d) - f[y(k), y(k-1), \cdots, y(k-n+1), u(k-1), \cdots, u(k-m+1)]}{g[y(k), y(k-1), \cdots, y(k-n+1), u(k-1), \cdots, u(k-m+1)]}.$$

де *d* – параметр передбачення повинен відповідати умові $d \geq 2$.

Загальна структура системи з регулятором NARMA-L2 має вигляд, наведений на рис. 8.52.

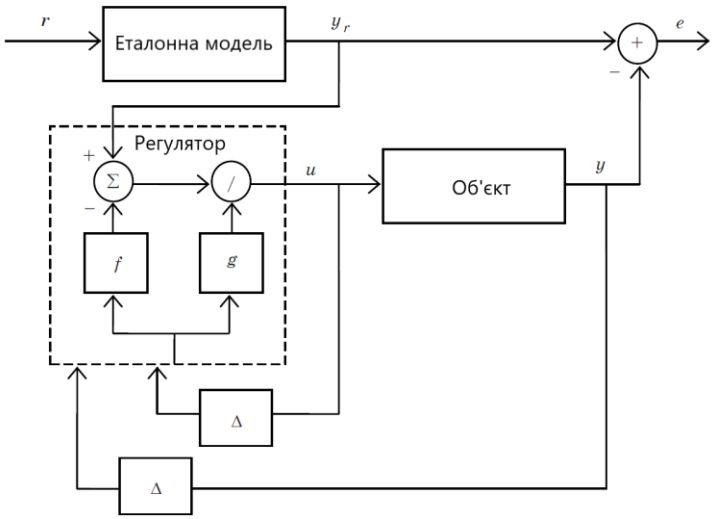


Рисунок 8.52 – Структурна схема регулятора на основі автомарегресії

На рисунку 8.53 показано реалізацію регулятора NARMA–L2.

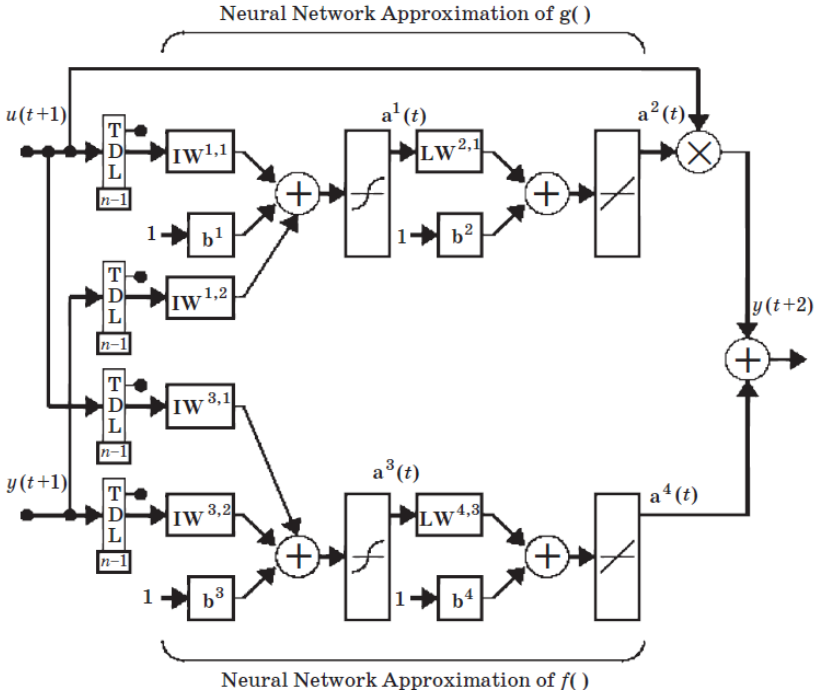


Рисунок 8.53 - NARMA–L2 регулятор в MatLab

Блок NARMA–L2 Controller знаходиться у бібліотеці інструментів MatLab у папці Neural Network Blockset/Control Systems

Приклад 8.5. Використання NARMA – L2-контролера

Розглянемо етапи проектування нелінійного регулятора NARMA – L2 на демонстраційному прикладі управління магнітною подушкою, реалізованого у пакеті Neural Network toolbox.

Приклад викликається командою narmamaglev (див рис. 8.54).

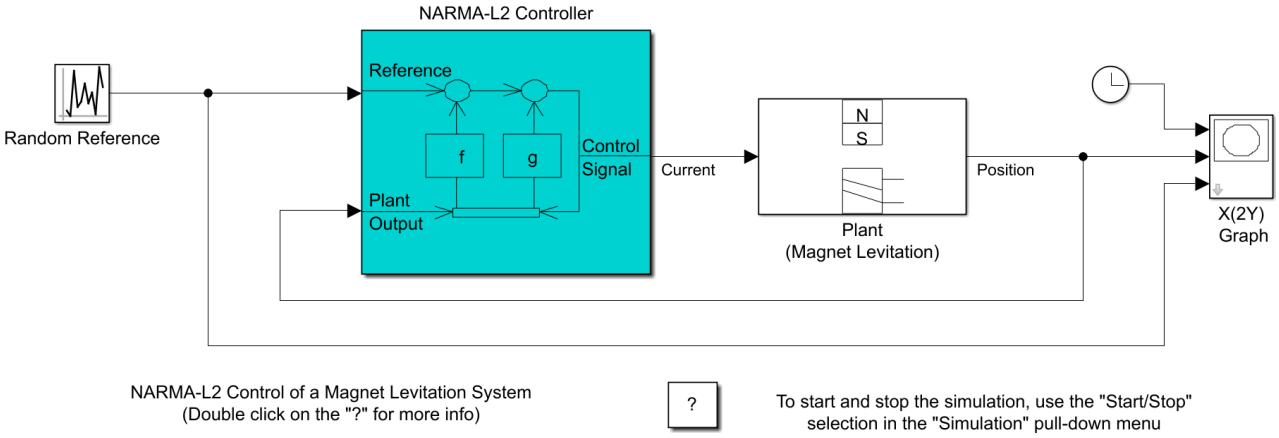


Рисунок 8.54 - Приклад використання NARMA – L2-контролера

До системи управління магнітною подушкою входять:

- блок керованого процесу Plant;
- блок контролера NARMA – L2 Controller;
- блок генерації еталонного ступінчастого сигналу з випадковою амплітудою Random Reference;
- блок відліку часу Clock;
- блок побудови графіків Graph.

У демонстраційному прикладі керованим об'єктом є постійний магніт, який рухається в магнітному полі тільки у вертикальному напрямку, як це показано схематично на рис. 8.55.

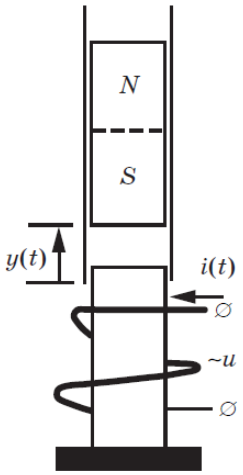


Рисунок 8.55 – Структура об'єкту управління

Рівняння руху постійного магніту має вигляд

$$\frac{d^2y(t)}{dt^2} = -g + \frac{\alpha}{M}\frac{i^2}{y(t)} - \frac{\beta}{M}\frac{dy(t)}{dt}.$$

де $y(t)$ - переміщення магніту, що відраховується від електромагніту; g – прискорення сили тяжіння; α – постійна, яка

залежить від числа витків обмотки та намагніченості сталевого сердечника; *i* - керуючий струм в обмотці електромагніту; *M* - маса постійного магніту.

Для початку роботи необхідно активізувати блок NARMA – L2 Controller подвійним натисканням лівої кнопки миші у вікні системи Simulink. Відобразиться вікно Plant Identification – NARMA – L2, параметри якого були розглянуті раніше у прикладі 5.4. Генерація навчальної послідовності проводиться за допомогою впливу ряду ступінчастих сигналів на математичну модель процесу управління та зняття значень на вході та виході моделі через кожен крок квантування. Графіки вхідного та вихідного сигналів відображаються у вікні Plant Input-Output Data (див. рис. 8.56).

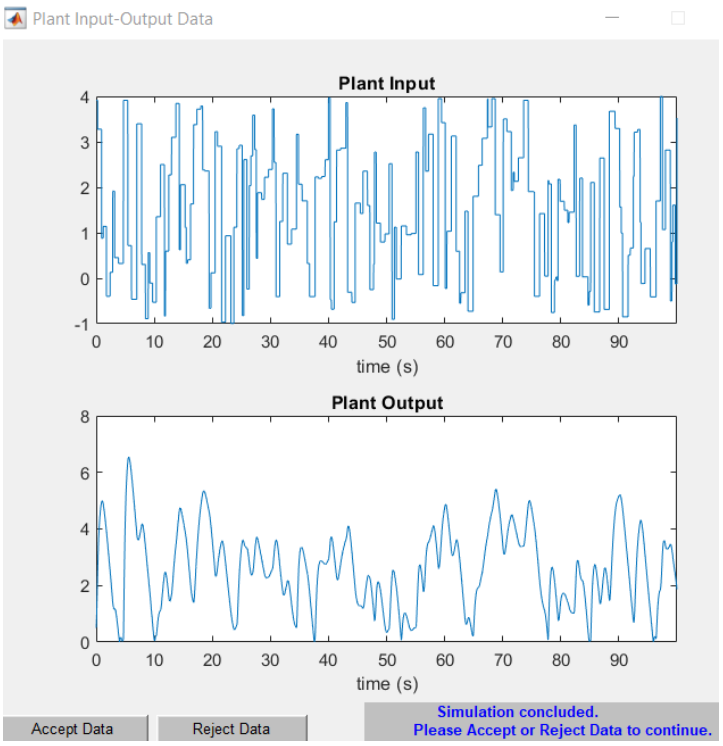


Рисунок 8.56 – Дані для навчання нейронної мережі

Після ідентифікації для навчання нейронної мережі скористайтеся кнопкою Train Network. Почнеться навчання нейромережевої моделі. По завершення навчання результати відображаються на графіках які наведені на рисунку 8.57.

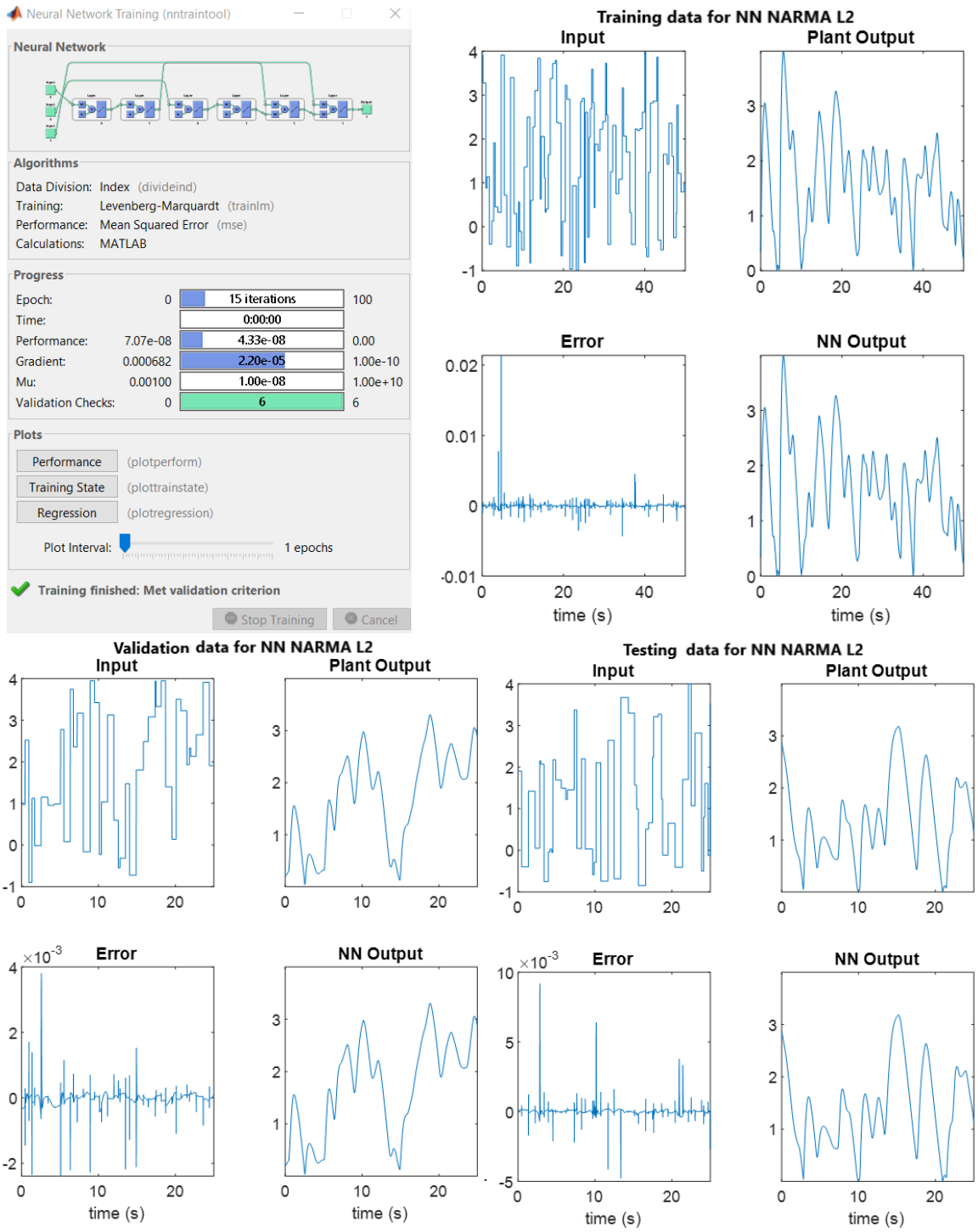


Рисунок 8.57 – Результати навчання NARMA – L2-контролера

Поточний стан зазначено у вікні Plant Identification повідомленням «Навчання завершено. Можна згенерувати або імпортувати нові дані, продовжити навчання або зберегти отримані результати, обравши кнопки OK або Apply». В результаті параметри нейромережевої моделі керованого процесу будуть введені до блоку NN NARMA – L2 Controller.

Вибравши опцію Start із меню Simulation, можна розпочати моделювання. У процесі математичного моделювання виводяться

графіки входу та виходу керованого процесу які наведені на рисунку 8.58.

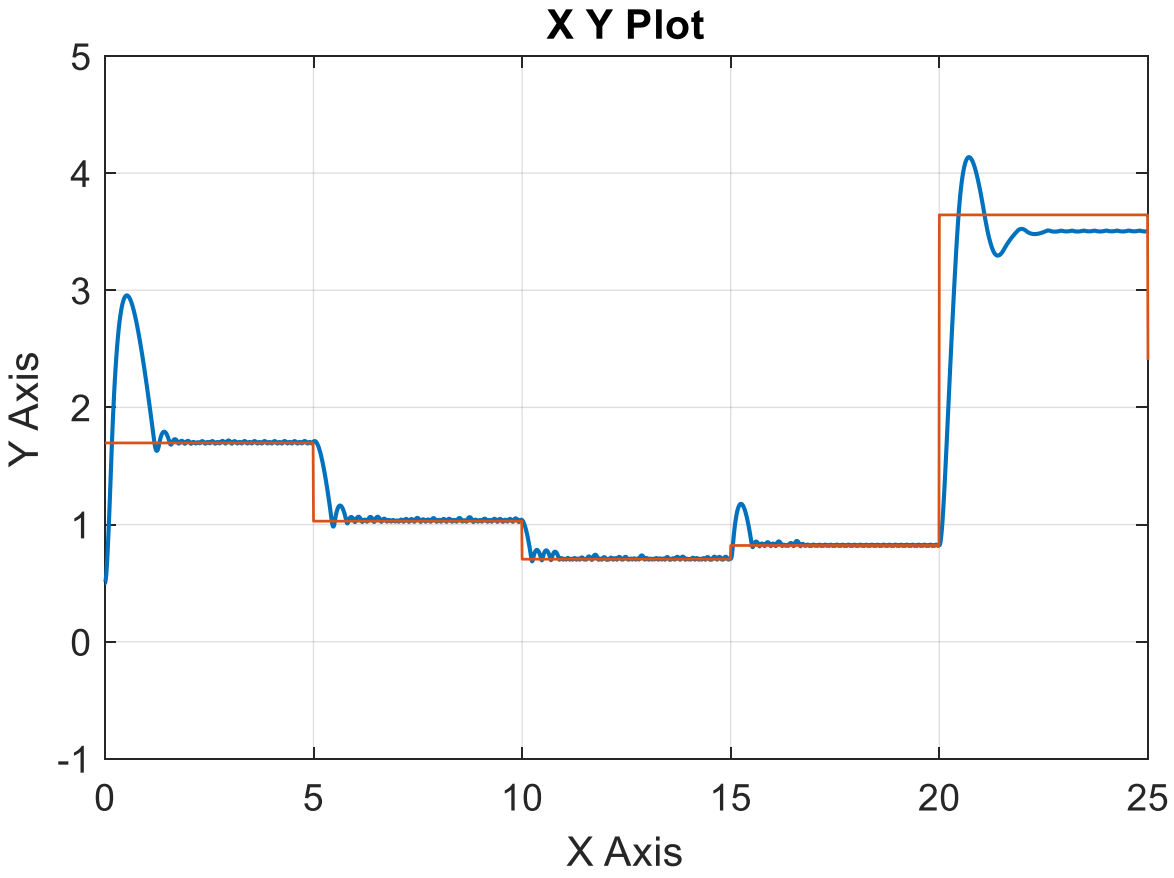


Рисунок 8.58 – Графікі входу та виходу керованого перехідного процесу

При керуванні на основі *еталонної моделі регулятора* – це нейронна мережа, яка має забезпечувати відстеження об'єктом заданого еталонного процесу. При цьому модель керуємого процесу активно використовується на етапі настройки параметрів регулятора. Необхідний обсяг обчислень можна порівняти з попереднім варіантом. Однак тут навчання засноване на динамічному варіанті зворотного розповсюдження помилки та є досить складним. Перевагою регуляторів на основі еталонної моделі є їхня застосовність до різних класів керованих об'єктів.

Проектування нейронних регуляторів з еталонною моделлю здійснюється у два етапи (див. рис. 8.59).

На першому етапі проводяться ідентифікація динамічної системи за допомогою наборів вхідних та відповідних їм вихідних величин, розробка архітектури нейронної моделі динамічної системи та налаштування її параметрів. Отримана ШНМ із заданою точністю відтворює поведінку динамічної системи і потім використовується для синтезу нейронного регулятора.

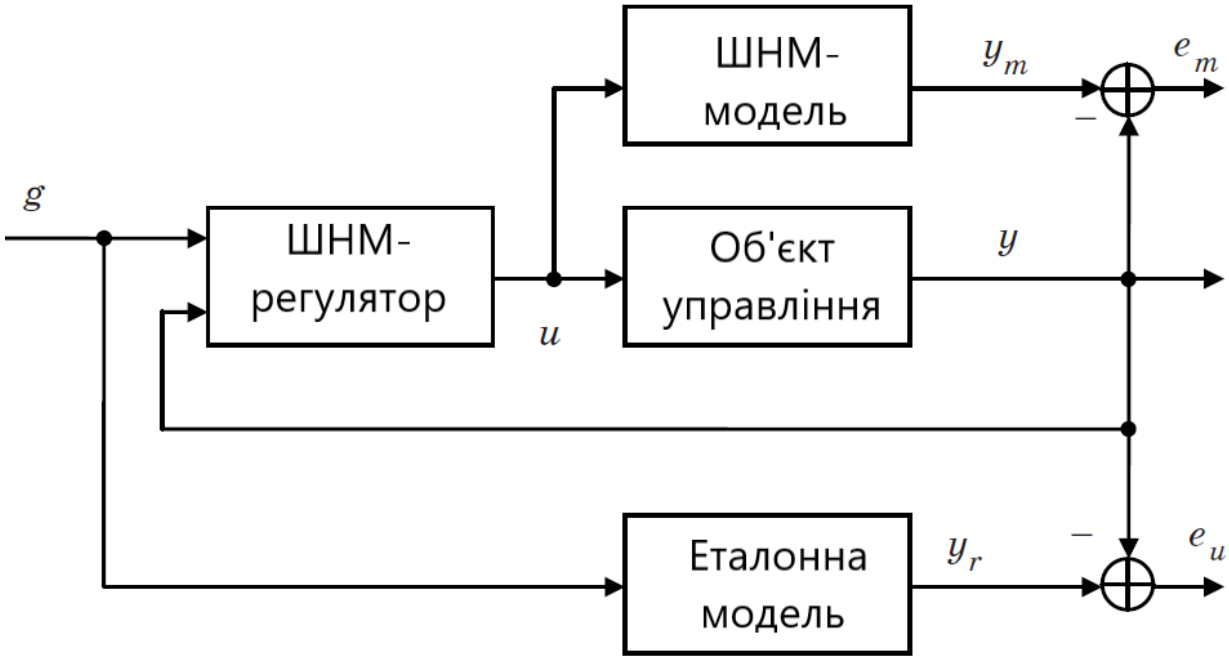


Рисунок 8.59 - Нейроуправління з еталонною моделлю

На другому етапі здійснюються пошук архітектури нейрорегулятора і його налаштування, щоб поведінка системи із заданою точністю відповідало поведінці еталонної моделі.

Блок Model Reference Controller також знаходиться у бібліотеці інструментів MATLAB у папці Neural Network Blockset/Control Systems.

Приклад 8.6. Нейроуправління з еталонною моделлю

Розглянемо проектування нейрорегулятора з еталонною моделлю для механічної ланки робота, динаміка якого описується нелінійним диференціальним рівнянням другого порядку з постійними коефіцієнтами:

$$\frac{d^2\varphi}{dt^2} = -2\frac{d\varphi}{dt} - 10\sin\varphi + u$$

де φ - кут повороту ланки; u - момент, що розвивається двигуном постійного струму.

Завдання нейрорегулятора полягає в тому, щоб за будь-якого зовнішнього впливу r система відтворювала із заданою точністю реакцію еталонної моделі, що описується диференціальним рівнянням

$$\frac{d^2y_r}{dt^2} = -6\frac{dy_r}{dt} - 9y_r + 9r$$

де y_r – вихід еталонної моделі.

Порівняємо перехідні процеси об'єкта та математичної моделі в MatLab (див. рис. 8.60 та 8.61).

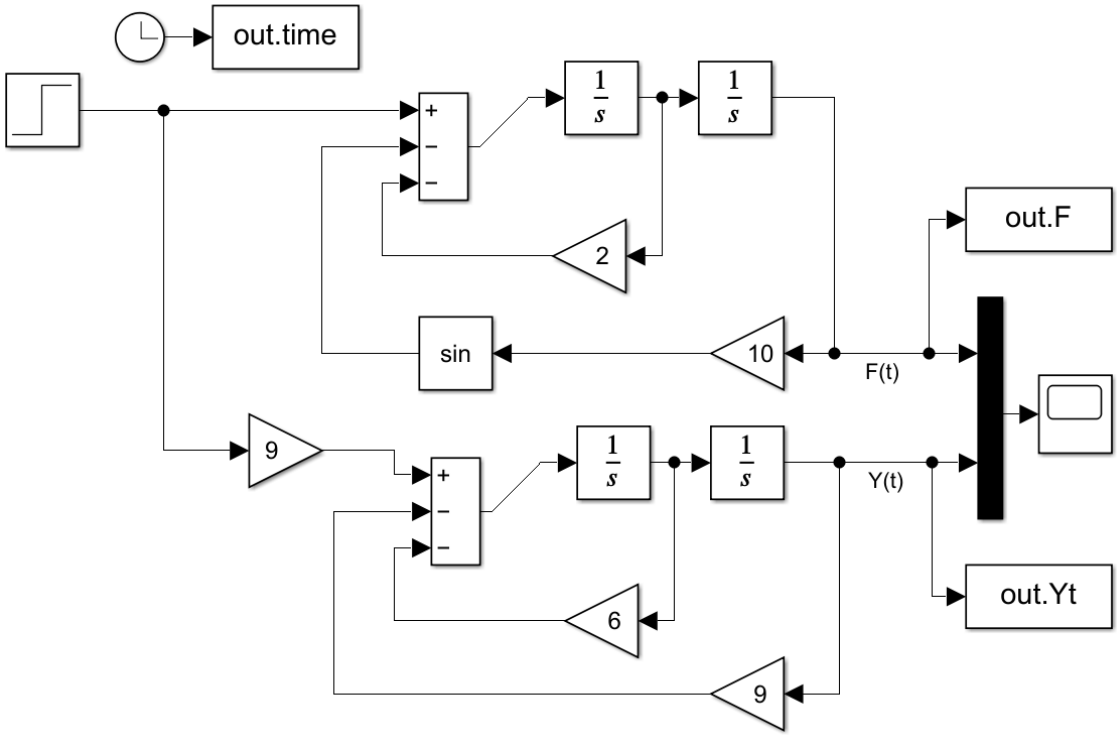


Рисунок 8.60 - Порівняння динаміки об'єкта та моделі в MatLab

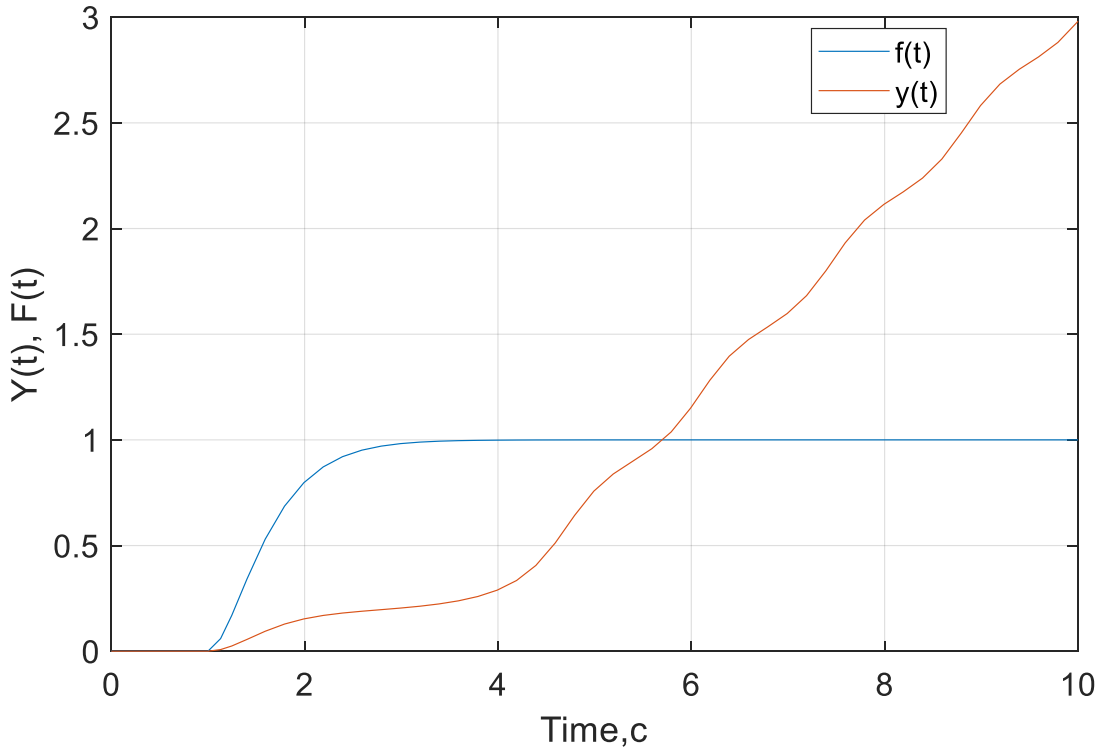


Рисунок 6.61 – Графіки перехідних процесів об'єкта та моделі

Аналіз графіків перехідних процесів наведених рисунку 8.61 показав, що, еталонна модель забезпечує стійкий перехідний процес, тоді як об'єкт управління без регулятора виявляється нестійким.

Розглянемо демонстраційний приклад, що ілюструє систему

управління з еталонною моделлю – ланкою маніпулятором робота. Приклад викликається командою mrefrobotarm (див. рис. 8.62).

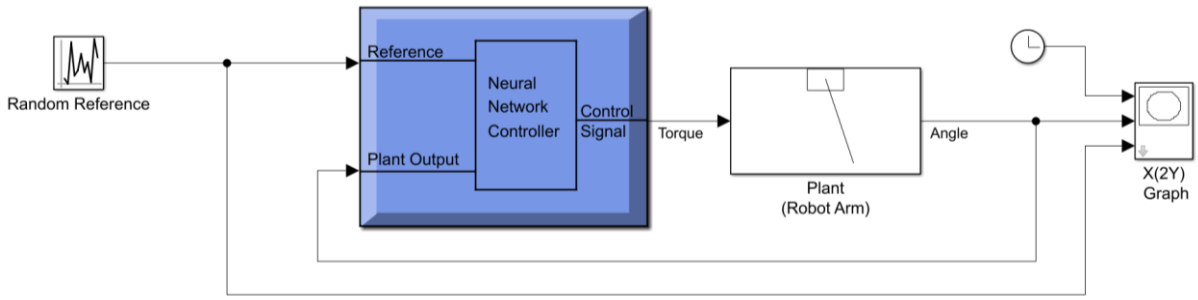


Рисунок 8.62 - Нейромережевий регулятор з еталонною моделлю

Для початку роботи з демонстраційним прикладом необхідно активізувати блок Model Reference Controller подвійним натисканням лівої кнопки миші. З'явиться вікно на рис. 8.63.

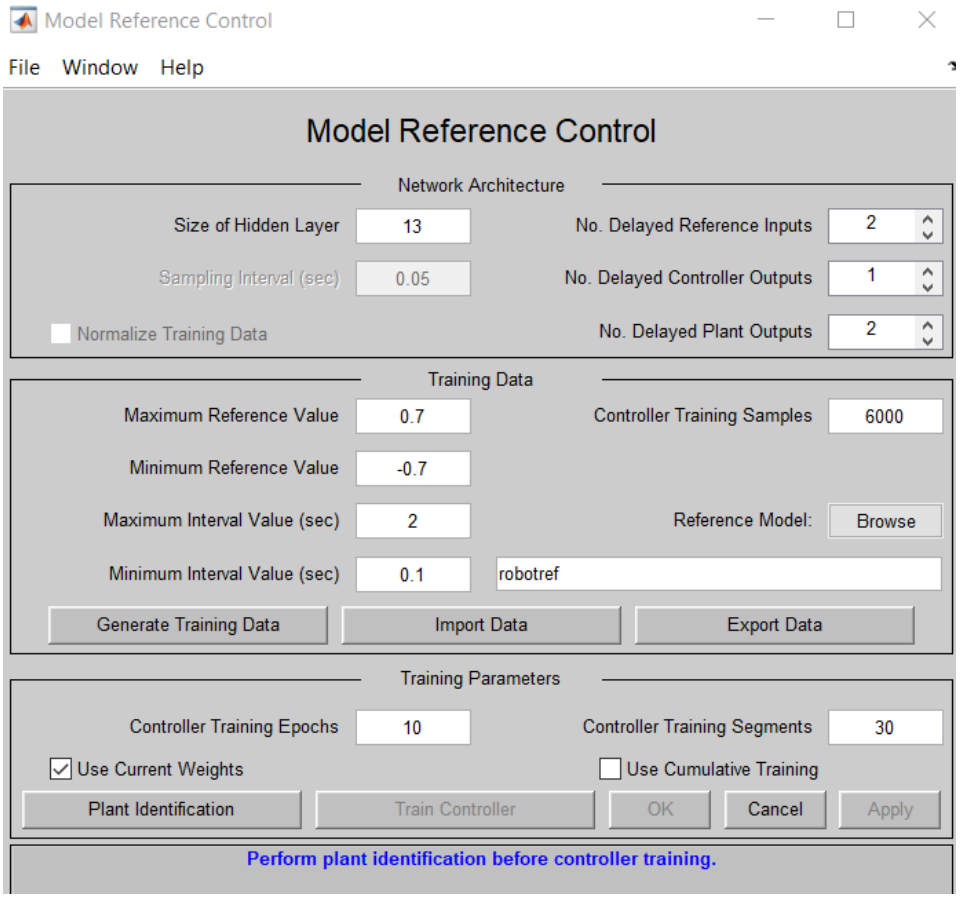


Рисунок 8.63 - Вікно налаштувань нейромережевої системи керування

Налаштування контролера здійснюється у два етапи. У першому етапі проводиться ідентифікація об'єкта управління, на другому – навчання нейроконтроллера.

1) Ідентифікація об'єкта управління здійснюється натисканням кнопки Plan Identification. У вікні Plan Identification (див. рис. 5.35) після налаштування (див. приклад 8.4) здійснюємо отримання даних навчання шляхом натискання кнопки Generate Training Data. Отримані результатів навчання після натискання кнопки Generate Training Data наведені на рисунку 8.64.

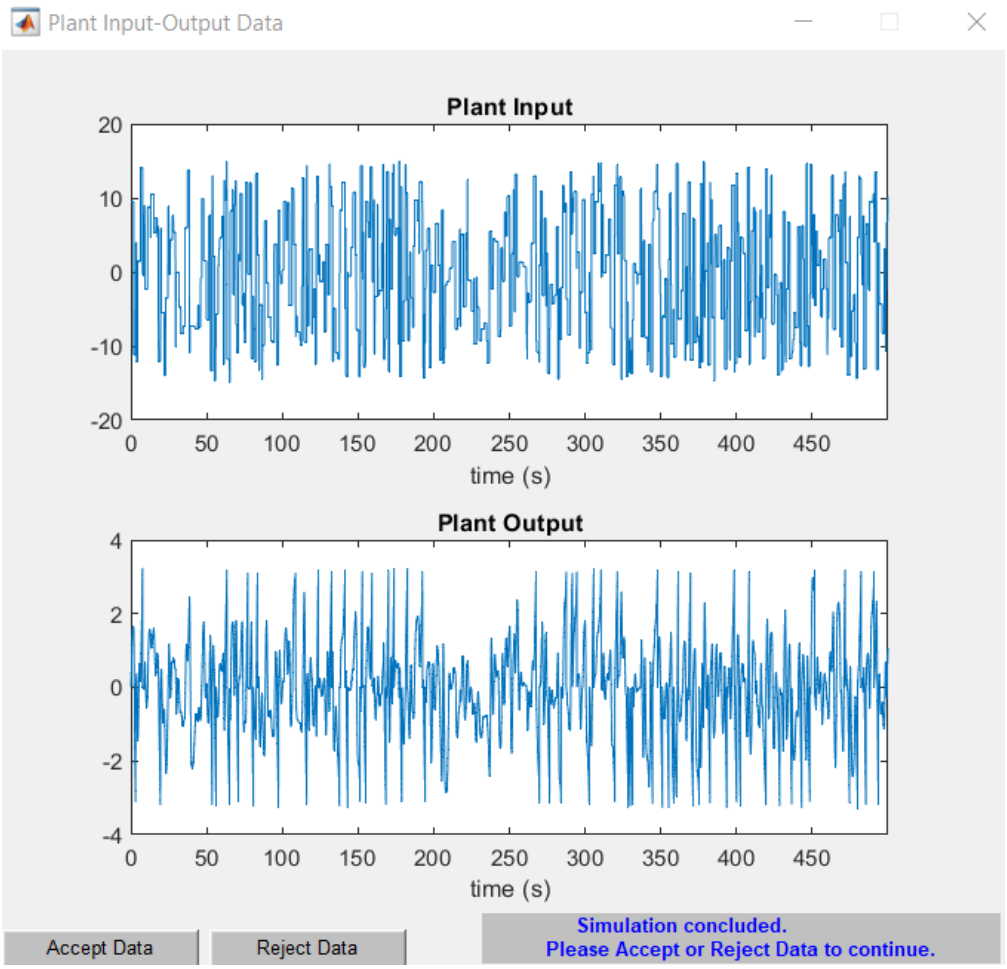


Рисунок 8.64 - Графіки вхідного та вихідного сигналів відображаються у вікні Plant Input-Output Data

Після завершення генерації навчальної послідовності необхідно або прийняти ці дані, натиснувши кнопку Accept Data, і тоді вони будуть використані для навчання нейронної мережі, або відкинути їх, натиснувши кнопку Reject Data.

Після отримання навчальної послідовності необхідно встановити необхідні параметри навчання та за допомогою кнопки Train Network (див. рис. 8.48) запустити процес навчання нейронної мережі. Після закінчення навчання його результати відображаються на графіках зміни помилки мережі для навчальної та тестової послідовності, а також у вікні вихідних значень моделі та мережі при подачі на вхід зазначених послідовностей.

Результати навчання НМ наведені на рисунку 8.65.

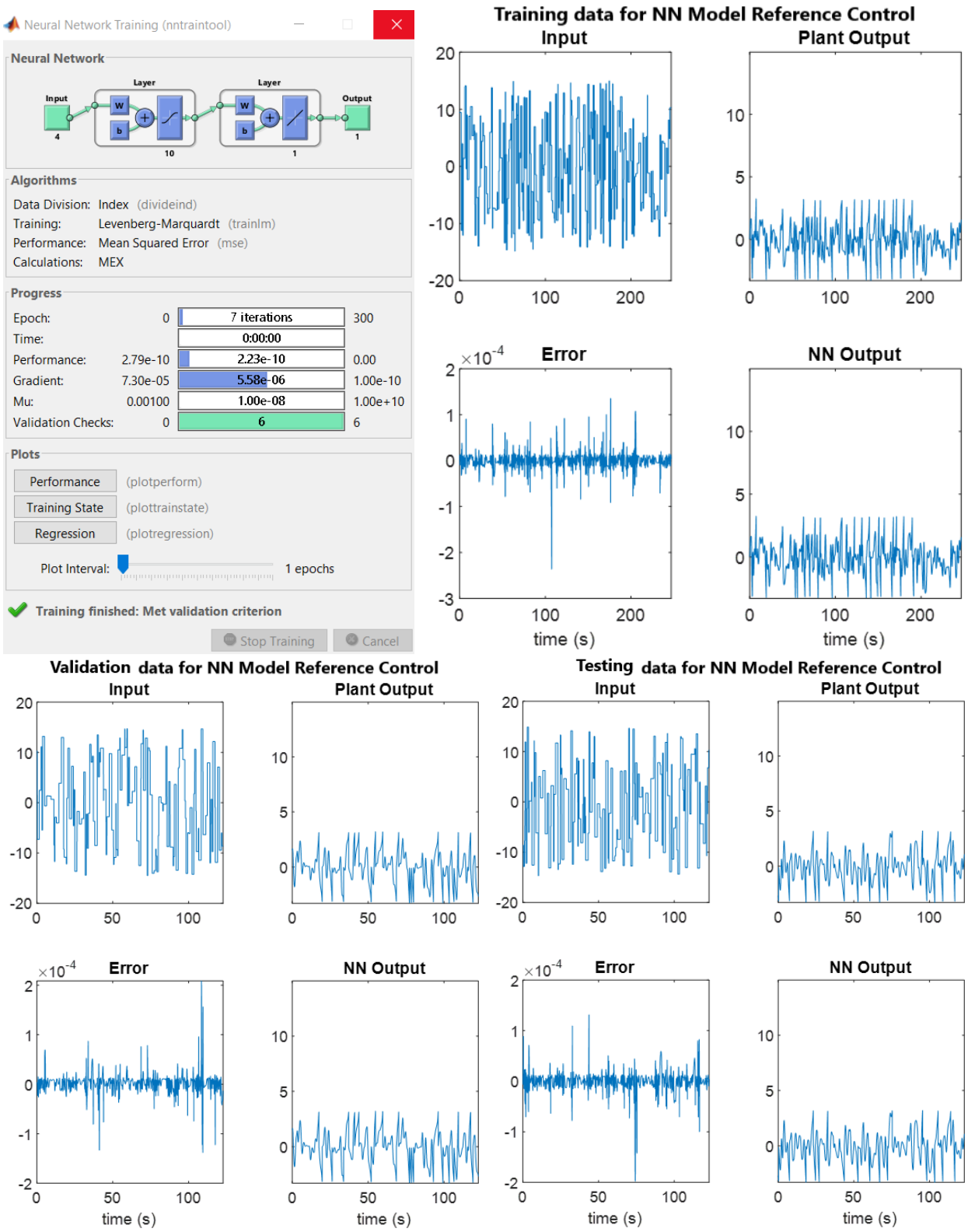


Рисунок 8.65 – Результати навчання нейронної мережі при ідентифікації об'єкта управління

У вікні Plan Identification приймаємо результати навчання і підтверджуємо послідовними натисканням кнопок Apply та OK.

Після цього переходимо к етапу навчання нейроконтролера у вікні Model Reference Controller.

2) Навчання нейроконтролера

Процес навчання нейроконтролера можна розбити три стадії:

- Перша стадія – вибір конфігурації керуючої мережі.

Тут параметри Sampling Interval та Normalize Training Data вибираються такими самими, як і під час навчання ідентифікаційної мережі автоматично.

No. Delayed Reference Inputs – кількість елементів затримки на вході, який подається тестовий сигнал.

No. Delayed Controller Outputs – кількість елементів затримки на виході керуючої мережі.

No. Delayed Plant Outputs – кількість елементів затримки на виході ідентифікаційної мережі.

- Друга стадія – отримання даних навчання.

На цій стадії, як і під час навчання ідентифікаційної мережі, можливе імпортування даних з mat-файлу або безпосередньо з робочого простору MatLab, а також даних на основі заздалегідь підготовленої еталонної моделі.

Reference model – це модель, яка визначає бажану поведінку замкнутої системи управління. Вона задається так же, як і модель об'єкта управління на етапі ідентифікації, тобто як моделі Simulink.

Controller Training Samples – число дискретних відкликів а тестовому сигналі. Воно має бути так ж, як і при ідентифікації об'єкта, досить великим.

Усі параметри, як і сам процес формування тестової послідовності, аналогічні використовуваним на етапі ідентифікації об'єкта управління.

Здійснюємо отримання даних навчання нейроконтролера шляхом натискання кнопки Generate Training Data. Отримані результатів навчання після натискання кнопки Generate Training Data наведені на рисунку 8.66.

Після завершення генерації навчальної послідовності необхідно або прийняти ці дані, натиснувши кнопку Accept Data, і тоді вони будуть використані для навчання нейронної мережі, або відкинути їх, натиснувши кнопку Reject Data.

- Третя стадія – навчання контролера (керівної мережі). Для навчання необхідно задати такі параметри:

Controller Training Epochs – кількість циклів навчання. Значення цього параметра не можна визначити заздалегідь, необхідно підібрати в процесі навчання.

Controller Training Segments – кількість сегментів, на які будуть поділені тестові дані. При цьому для кожного сегмента буде проведено вказану кількість циклів. Бажано розбивати дані на сегменти таким чином, щоб кожен сегмент містив хоча б один перехідний процес.

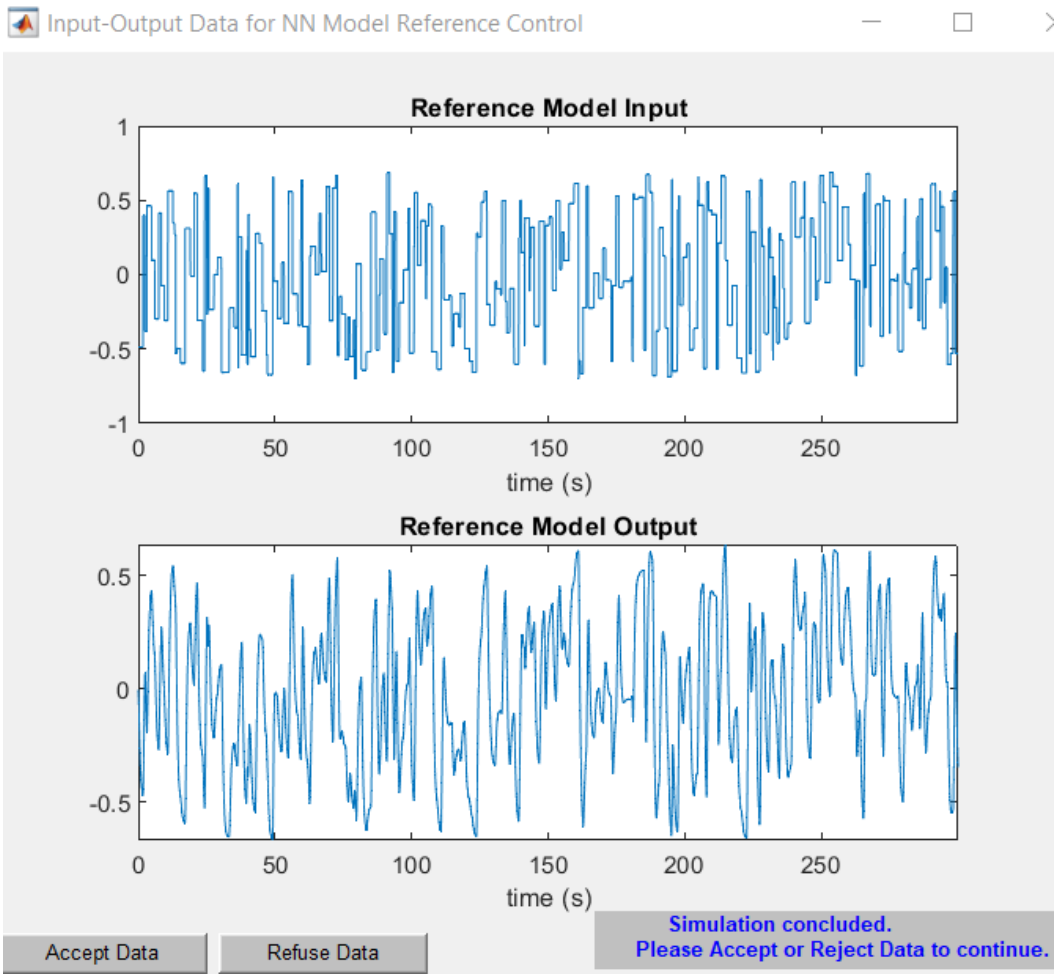


Рисунок 8.66 - Графіки вхідного та вихідного сигналів відображаються у вікні Input-Output Data for NN Model Reference Control

Use Current Weights – використовувати поточні вагові коефіцієнти. В іншому випадку ці коефіцієнти будуть обрані випадковим чином. Цей параметр слід застосовувати під час повторного навчання.

Use Cumulative Training – при виборі цього параметра початкове навчання буде виконано з першим сегментом тестових даних, а при подальшому навчанні інші сегменти додаватимуться до даних, що використовуються на попередньому етапі навчання. Таким чином, на фінальній стадії навчання буде використано весь набір тестових даних. Цей параметр слід застосовувати обережно, оскільки він підвищує час навчання.

Після завдання всіх параметрів необхідно розпочати навчання керуючої мережі, для чого необхідно натиснути кнопку Train controller. При цьому відкриється вікно, в якому відображатиметься процес навчання (так само, як і під час навчання ідентифікаційної мережі). Після закінчення навчання відкривається вікно Plant Response for NN Model Reference Control, де відображаються результати навчання (див. рис 8.67).

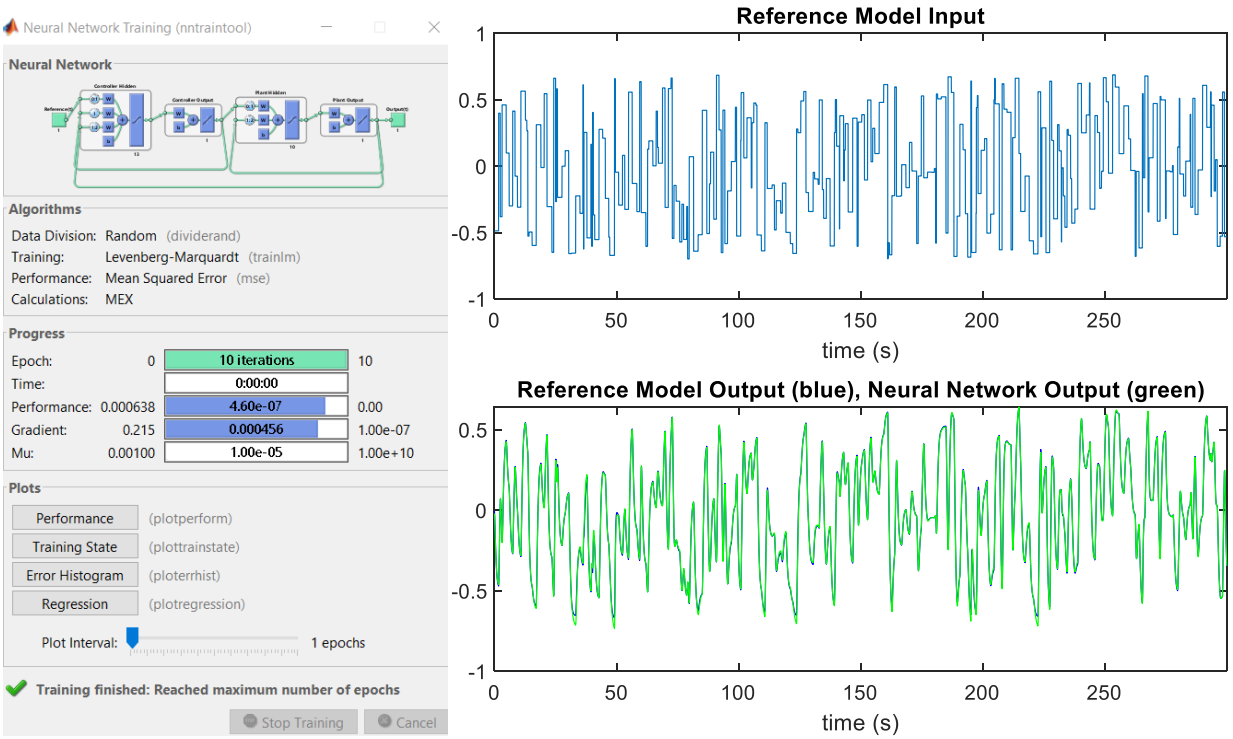


Рисунок 8.67 - Результати навчання нейроконтролера

Після успішного завершення процесу навчання керуючої мережі контролер готовий до використання у складі системи керування. На рисунку 8.68 наведено приклад відпрацювання контролером заданого вхідного ступінчастого сигналу.

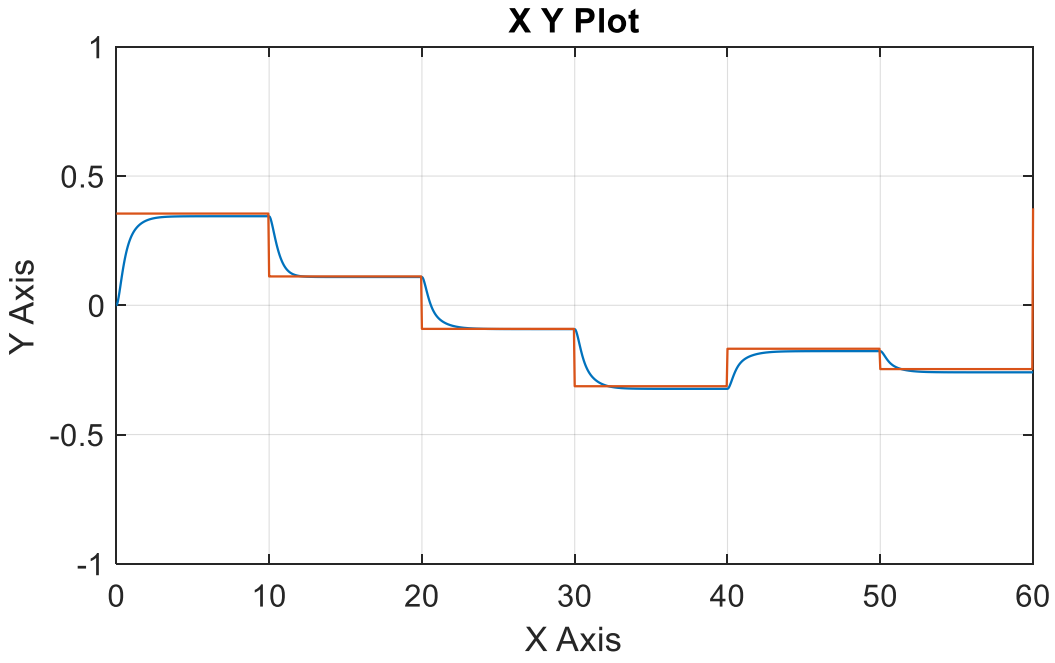


Рисунок 5.54 - Відпрацювання системою управління еталонного сигналу

Блок Model Reference Controller можна використовувати для довільного об'єкта управління. При цьому можуть виникати такі проблеми:

1. Процес навчання ідентифікаційної мережі йде повільно або припиняється мимовільно, що визначається тому, що помилка навчання не зменшується з будь-якої епохі навчання. Це може статися з таких причин:

- невірно обрана конфігурація ШНМ – необхідно задати більшу кількість нейронів у прихованому шарі;
- неправильно заданий набір тестових даних, що не містить достатньої для навчання кількості інформації;
- вибрано дуже малу кількість епох навчання, і їх необхідно збільшити.

2. Процес навчання керуючого контролера йде повільно, і помилка навчання мало зменшується.

Це може відбуватися у таких випадках:

- невдало задані параметри тестової послідовності при ідентифікації об'єкта, тоді необхідно повторити процес навчання мережі з іншими параметрами у блоці Training Data; слід використовувати близькі за формою та параметрами тестові сигнали під час навчання обох ШНМ;
- вибрано занадто малу кількість елементів у прихованому шарі, і його необхідно збільшити.

3. Прямі показники якості системи, що моделюється, істотно відрізняються від аналогічних показників для еталонної моделі.

Це можливо у таких випадках:

- ідентифікація об'єкта здійснена невдало;
- конфігурація керуючої мережі не дозволяє вирішувати завдання керування, і необхідно повторити навчання зі збільшеною кількістю елементів у прихованому шарі.

9 НЕЧІТКІ РЕГУЛЯТОРИ В СИСТЕМАХ АВТОМАТИЗАЦІЇ

9.1 Ковзаючий режим нечіткого регулятора

Розглянемо відображення таблиці лінгвістичних змінних на фазову площину при 5 термах, що описують кожну змінну (рис. 9.1).

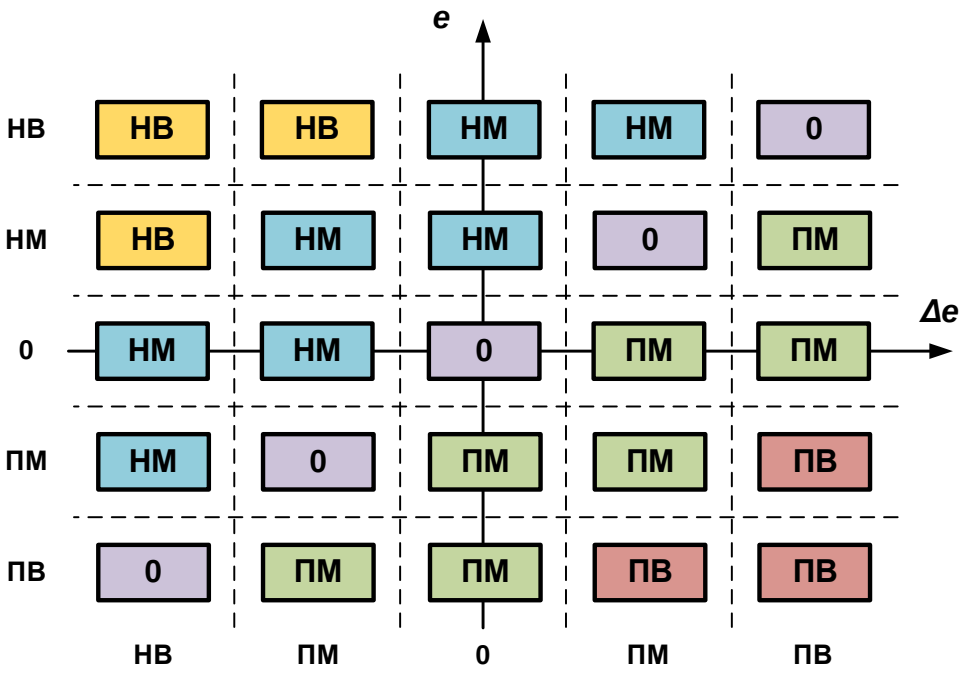


Рисунок 9.1 – Відображення закону управління фазову площину

Виділимо на фазовій площині області, де управління однаково (див. рис. 9.2).

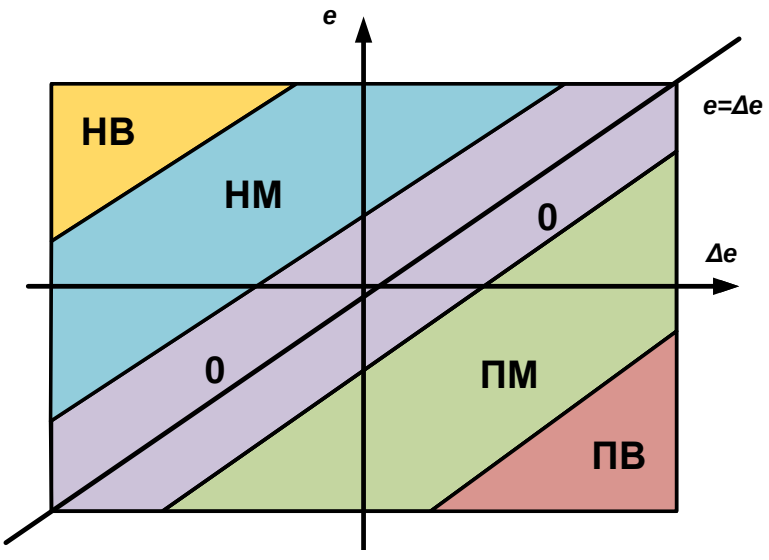


Рисунок 9.2 - Різні області сигналу керування на фазовій площині

Як свідчить рис. 9.2 величина модуля сигналу управління пропорційна відстані від прямої $e = \Delta e$. На використанні цієї особливості заснований так званий ковзний режим нечіткого регулятора.

Ковзаючий режим управління – робастний метод управління нелінійними системами в умовах невизначеності, який застосовується для об'єктів без істотного запізнення.

Розглянемо нелінійний об'єкт з одним входом та одним виходом (SISO):

$$x^{(n)} = f(X) + u;$$

$$X = \left(x, \dot{x}, \dots, x^{(n)}\right)^T,$$

де $u \in R$ - керуючий вхід; $x \in R$ – вихід, $X \in R^n$ – вектор стану.

Невизначеність моделі описується функцією

$$f(X) = \hat{f}(X) + \Delta f(X);$$

$$\Delta f(X) \le F(X)$$

де оцінка $\hat{f}(X)$ та $F(X)$ - відомі; $\Delta f(X))$ – невідома величина.

Мета управління полягає у виробленні такого сигналу зворотного зв'язку $u = u(X)$, щоб стан системи X наближався до бажаного стану X_d, тобто.

$$e = X - X_d \to 0.$$

Визначимо скалярну функцію

$$S(X,t) = \left(\frac{d}{dt} + \lambda\right)^{n-1} e = e^{(n-1)} + C_{n-1}^1 \lambda e^{(n-2)} + C_{n-1}^2 \lambda^2 e^{(n-3)} + \cdots + \lambda^{(n-1)} e$$

де λ – позитивна константа. Рівняння

$$s(X,t) = 0 \qquad (9.1)$$

визначає залежну від часу перемикальну поверхню в просторі станів R^n.

Для системи другого порядку (n = 2) поверхня ковзання $S(t)$ визначається виразом

$$S(X,t) = \dot{e} + \lambda e = \dot{x} + \lambda x - \dot{x_d} - \lambda x_d = 0$$

Рівняння (9.1) має єдине рішення $e(t) = 0$ за нульових початкових

умов $e(0) = 0$.

Проблема керування траєкторією зводиться до завдання забезпечення (9.1).

Це може бути забезпечено при виконанні умови ковзання:

$$\frac{1}{2}(s)^2 \leq -\eta|s|$$

де η - позитивна константа.

Інакше кажучи, для того щоб у системі існував ковзний режим, необхідно сформувати управління таким чином, щоб точка, що зображає, один раз потрапивши на поверхню ковзання, вже не могла зійти з неї і далі рухалася тільки вздовж цієї поверхні. Нехай у деякий момент часу $S > 0$. Тоді похідна за часом dS/dt (швидкість зміни S) має бути негативною. У цьому випадку S (відхилення від поверхні ковзання) буде зменшуватися до нуля, тобто зображувальна точка буде рухатися у напрямку до S. Нехай, далі точка перетину поверхню $S = 0$ і тепер $S<0$. Відповідно, повинно змінитися і значення похідної, а саме виконати умову $dS/dt > 0$. У цьому випадку зображувальна точка буде рухатися назад у напрямку до поверхні ковзання.

Відзначимо лише, що режим ковзання вимагає опису фазової траєкторії бажаної системи (лінії ковзання). Якщо реальна фазова траєкторія перетинає бажану фазову траєкторію, сигнал управління перемикається. Перемикання можуть відбуватися з дуже великою (теоретично нескінченною) частотою, утримуючи при цьому перехідний процес на лінії ковзання. При зміні властивостей об'єкта змінюється лише частота перемикання сигналу управління, а траєкторія, вздовж якої проходить рух, залишається незмінною.

Таким чином, можна вважати, що задана лінія перемикання, де значення сигналу управління дорівнює нулю. Вище та нижче цієї лінії сигнал керування має різні знаки, а його модуль пропорційний відстані від цієї лінії (рис. 9.2):

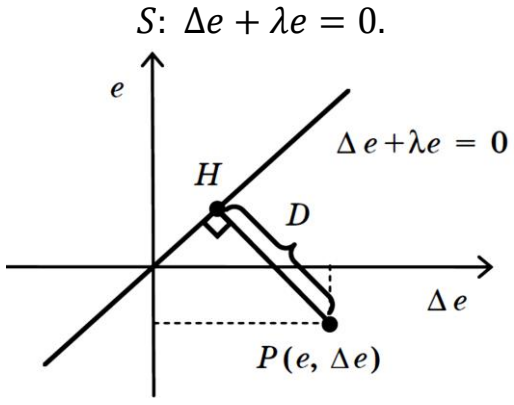


Рисунок 9.2 - Відстань до лінії перемикання

Розглянемо розрахунок дистанції від точки у фазовому просторі до лінії перемикання (рис. 9.2 де *H* – точка перетину лінії перемикання і перпендикуляра, проведеного до цієї лінії з точки *P*).

Відстань розраховується за відомими формулами (замість похідної тут використано збільшення):

$$d = \sqrt{(e - e_1)^2 + (\Delta e - \Delta e_1)^2} = \frac{|\Delta e_1 + \lambda \Delta e_1|}{\sqrt{1 + \lambda^2}}.$$

З урахуванням знаку відстань можна описати формулою

$$D = sgn(S)\frac{|\Delta e_1 + \lambda \Delta e_1|}{\sqrt{1 + \lambda^2}} = \frac{\Delta e_1 + \lambda \Delta e_1}{\sqrt{1 + \lambda^2}} \qquad (9.2),$$

де

$$sgn(S) = \begin{cases} 1, & \text{якщо } S > 0; \\ -1, & \text{якщо } S < 0\,. \end{cases}$$

Таким чином, за рахунок використання відстані до лінії перемикання зі знаком можна описати нечіткий закон управління одномірною таблицею (табл. 9.1).

Таблиця 9.1 – Одномірний нечіткий закон управління

D	x_1	x_2	x_3	x_4	x_5
*D**	НВ	НМ	0	ПМ	ПВ
*U**	НВ	НМ	0	ПМ	ПВ

У табл. 9.1 величина *D** визначає лінгвістичне значення відстані, а *U** – лінгвістичне значення сигналу управління. Число правил, що описують нечіткий закон управління, скорочується таким чином у п'ять разів (порівняно з таблицями нечітких правил синтезу нечітких ПД та ПІ - регуляторів). У цьому полягає основна перевага використання ковзного режиму НЛР. Справді, величини x_i, зазначені у табл. 9.1 є центрами відповідних термів, і нечіткий закон управління вимагає завдання всього 5 параметрів, а не 25.

Лінію перемикання можна описати і у випадку, коли на вході регулятора подається більше двох вхідних змінних. В загальному випадку перемикальна поверхня задається рівнянням

$$S = e^{(n-1)} + \lambda_{n-1}\lambda e^{(n-1)} + \cdots + \lambda_2 \dot{e} + \lambda_1 e = 0$$

Відповідно, відстань до лінії перемикання визначається за формулою

$$D = \frac{e^{(n-1)} + \lambda_{n-1}\lambda e^{(n-1)} + \cdots + \lambda_2 \dot{e} + \lambda_1 e}{\sqrt{1 + \lambda_{n-1}^2 + \cdots + \lambda_2^2 + \lambda_1^2}}.$$

Отже, змінюється лише формула обчислення відстані, а алгоритм нечіткого управління залишається тим самим, і завдання проектування НЛР завжди значно спрощується.

Розглянемо приклад моделювання роботи НЛР у режимі спостереження. Нехай об'єкт управління заданий передавальною функцією

$$W = \frac{6{,}5}{0{,}02p^2 + 0{,}4p + 1}$$

На рис. 9.4 представлена схема моделювання в MatLab Simulink.

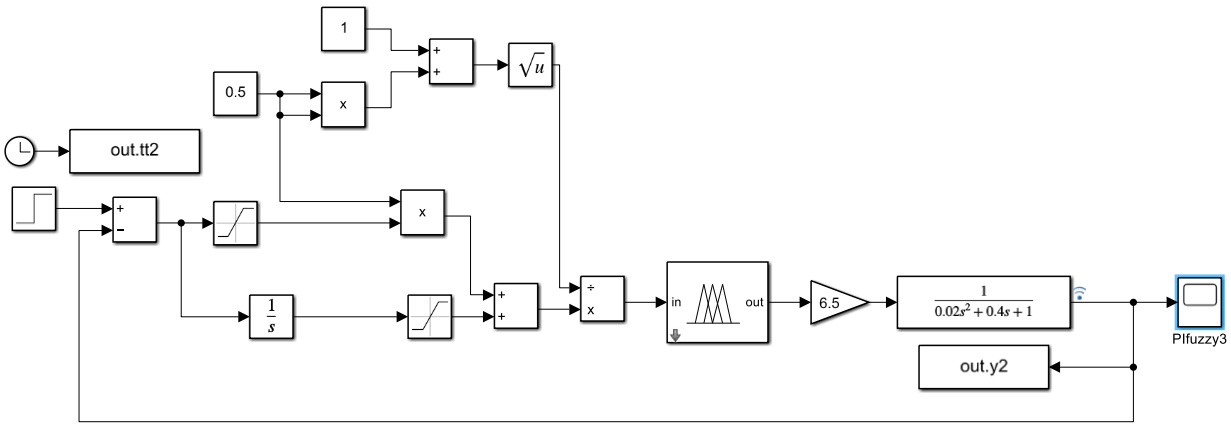


Рисунок 9.4 – Математична модель НЛР в режимі спостереження в MatLab Simulink

Вхідний сигнал НЛР формується відповідно до (9.5). Нечіткий закон управління відповідає табл. 9.8. При рівномірному розташуванні термів вхідно-вихідних лінгвістичних змінних завдання конструювання НЛР зводиться до правильного вибору коефіцієнта λ.

Синтезуємо Fuzzy Logic Controller *НЛР в режимі спостереження* надаємо Fis name: PISfis7 (ім'я може буди будь-яке).

Викликається редактор системи нечіткого висновку (Fuzzy Inference System – FIS) у MatLab командою

>> fuzzy

Дале проводиться налаштовування головного вікна редактора нечіткої логічної системи (див. рис. 9.5.).

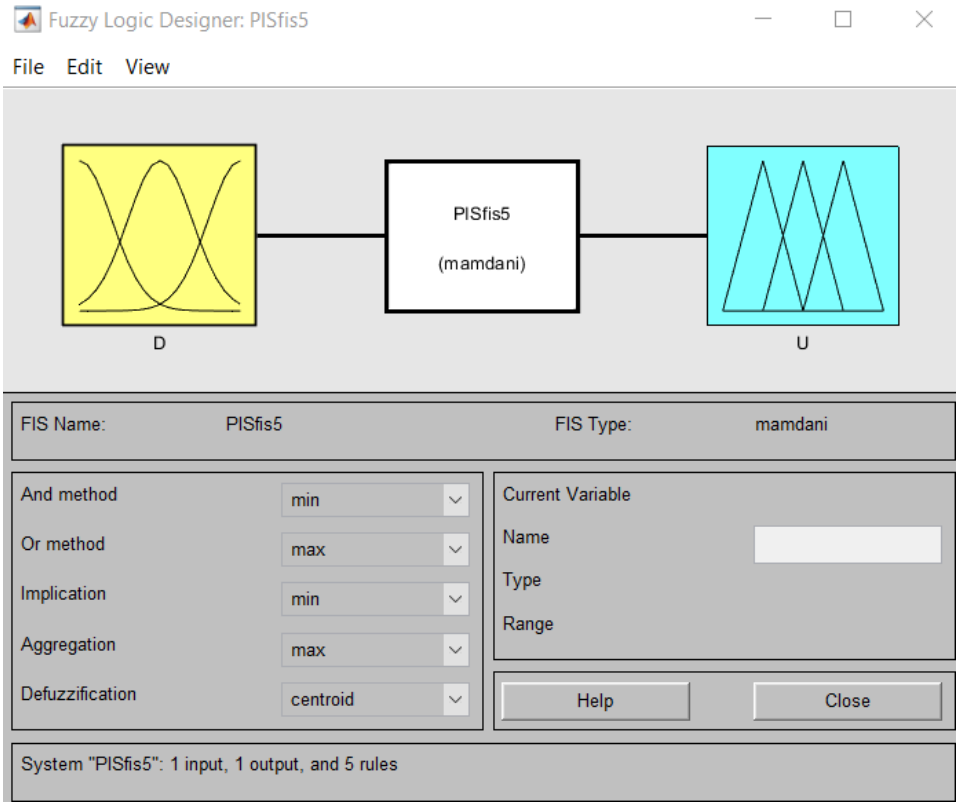


Рисунок 9.5 – Налаштування інтерфейсу FIS editor

У *FIS editor* задається опис систему нечіткого логічного висновку *Mamdani*. Для створюваної системи обирається вид логічного зв'язку (*And method – min*) та (*Or method – max*), вид імплікації (*Implication – min*), спосіб агрегування висновків правил (*Aggregation – max*) та метод дефазифікації (*Defuzzification – centroid*).

У меню *Edit* послідовно додаємо вхідну змінну з розміром базової шкали (Range= [-1 1]) та вихідну змінну з розміром базової шкали (*Range*= [-1 1]).

Для опису вхідних логічних змінних у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної змінної трикутну функцію приналежності.

Терми лінгвістичної змінної «Помилка управління» розміщуються відповідно:

Name = "NB"; Type = "trimf"; Param = [-1.33 -1 -0.33];
Name = "NM"; Type = " trimf"; Param = [-1 -0.33 -0.03];
Name = "NS"; Type = " trimf"; Param = [-0.33 -0.033 0];
Name = "Z"; Type = " trimf"; Param = [-0.1 0 0.1];
Name = "PS"; Type = " trimf"; Param = [0 0.033 0.33];
Name = "PM"; Type = " trimf"; Param = [0.033 0.33 1];
Name = "PB"; Type = "trimf"; Param = [0.33 1 1.33].

Результат налаштування функцій приналежності вхідних змінних «Помилка управління» наведено на рис. 9.6.

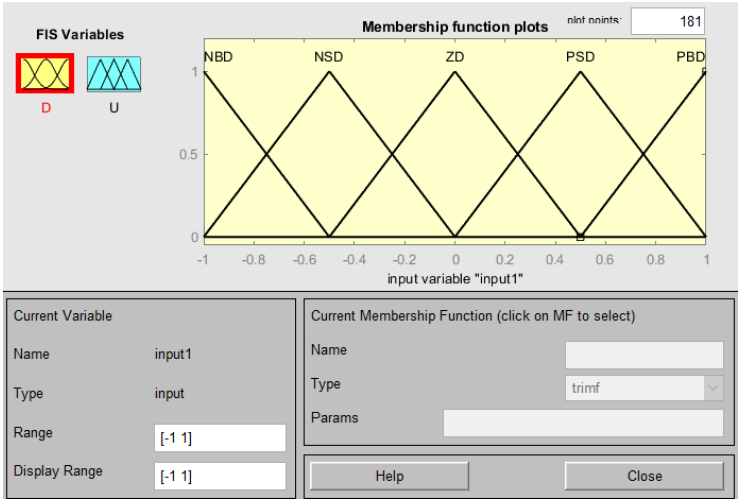


Рисунок 9.6 – Результат налаштування функцій приналежності вхідних змінних «Помилка управління»

Для опису вихідних логічних змінних у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної вихідної змінної трикутну функцію приналежності. З параметрами:

Name = "NBD"; Type = "trimf"; Param = [-1.5 -1 -0.5];
Name = " NSD "; Type = " trimf"; Param = [-1 -0.5 0];
Name = "ZD"; Type = " trimf"; Param = [-0.5 0 0.5];
Name = "PSD"; Type = " trimf"; Param = [0 0.5 1];
Name = "PBD"; Type = "trimf"; Param = [0.5 1 1.5].

Результат налаштування функцій приналежності вхідних змінних наведено на рис. 9.7.

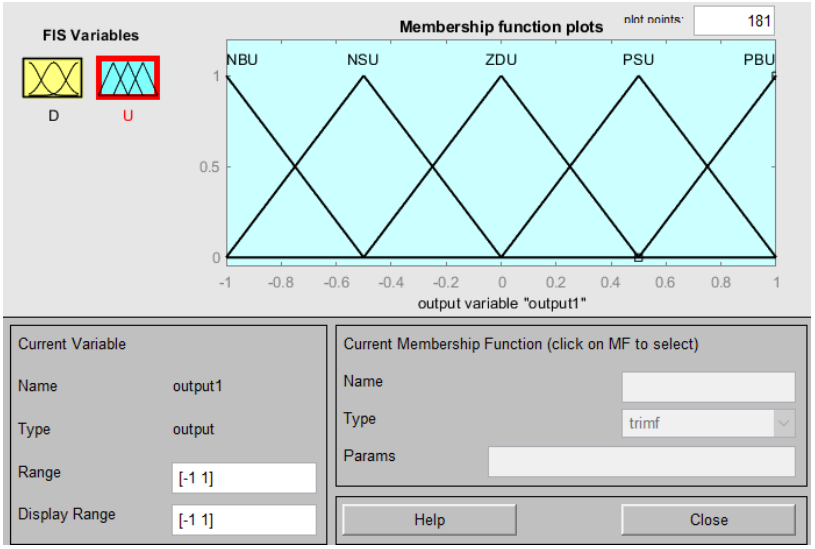


Рисунок 9.7 – Результат налаштування функцій приналежності вихідних змінних

Після опису вхідні та вихідні лінгвістичні змінні, здійснюється опис правил у вікні редактора Rule Editor, кількість правил для даного випадку дорівнює 5. Управляючі правила сформуємо згідно з табл. 9.8:

1. if (input1 is NBD) then (output1 is NBu) (1)
2. if (input1 is NSD) then (output1 is NSu) (1)
3. if (input1 is ZD) then (output1 is Zu) (1)
4. if (input1 is PSD) then (output1 is PSD) (1)
5. if (input1 is PBD) then (output1 is PBU) (1)

Налаштовування управляючих правил для даного випадку наведено на рисунку (див. рис. 9.8).

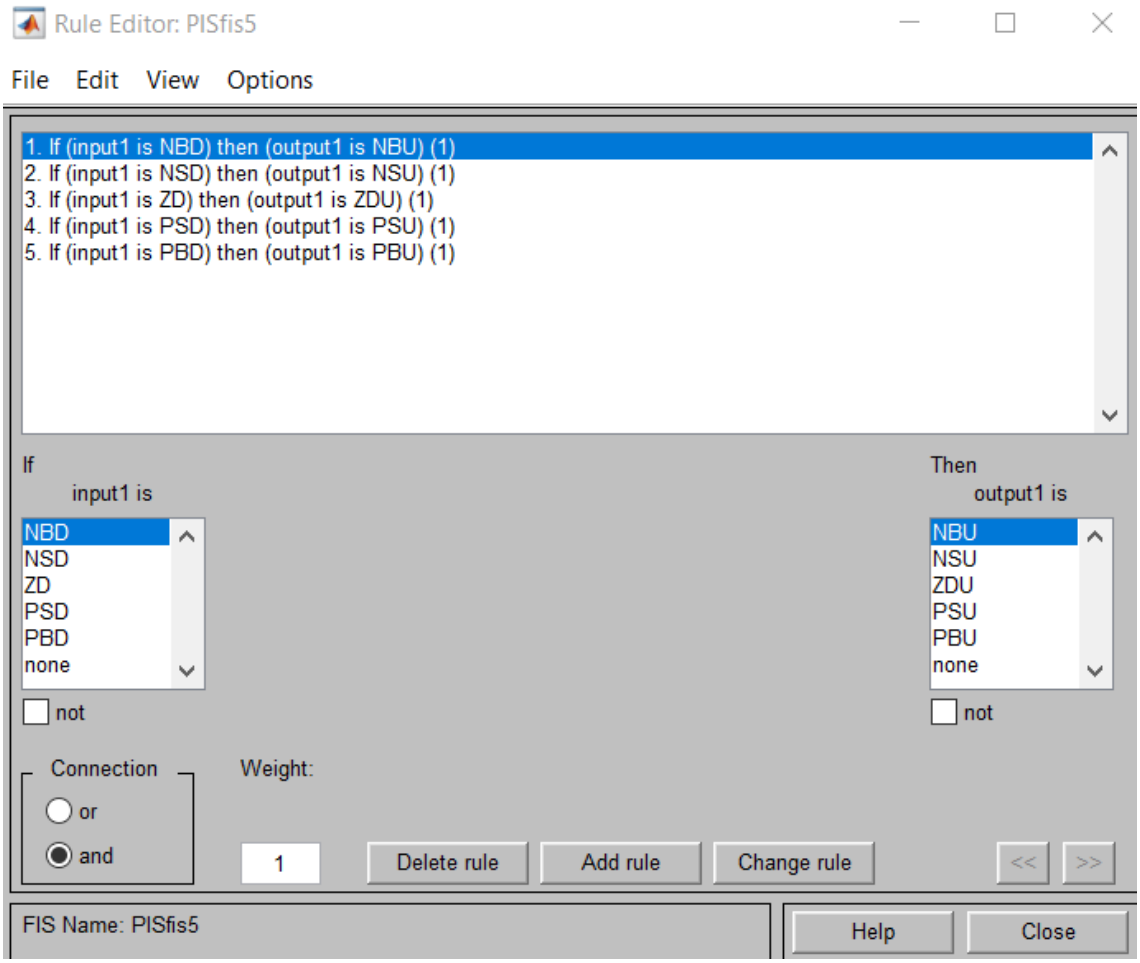


Рисунок 9.8 - Налаштовування опису правил функціонування НЛР

Посилки у правилах пов'язані (connection) за допомогою операції *and*. Введене правило забезпечене ваговим коефіцієнтом (*Weight*=1).

Після опису правил можливо за допомогою

- інтерфейсу *Rule Viewer* можливо переглянути роботу системи нечіткого висновку за різних вхідних даних (див. рис. 9.9);

− інтерфейсу Surface Viewer можливо переглянути графічне подання закону управління (рис. 9.10).

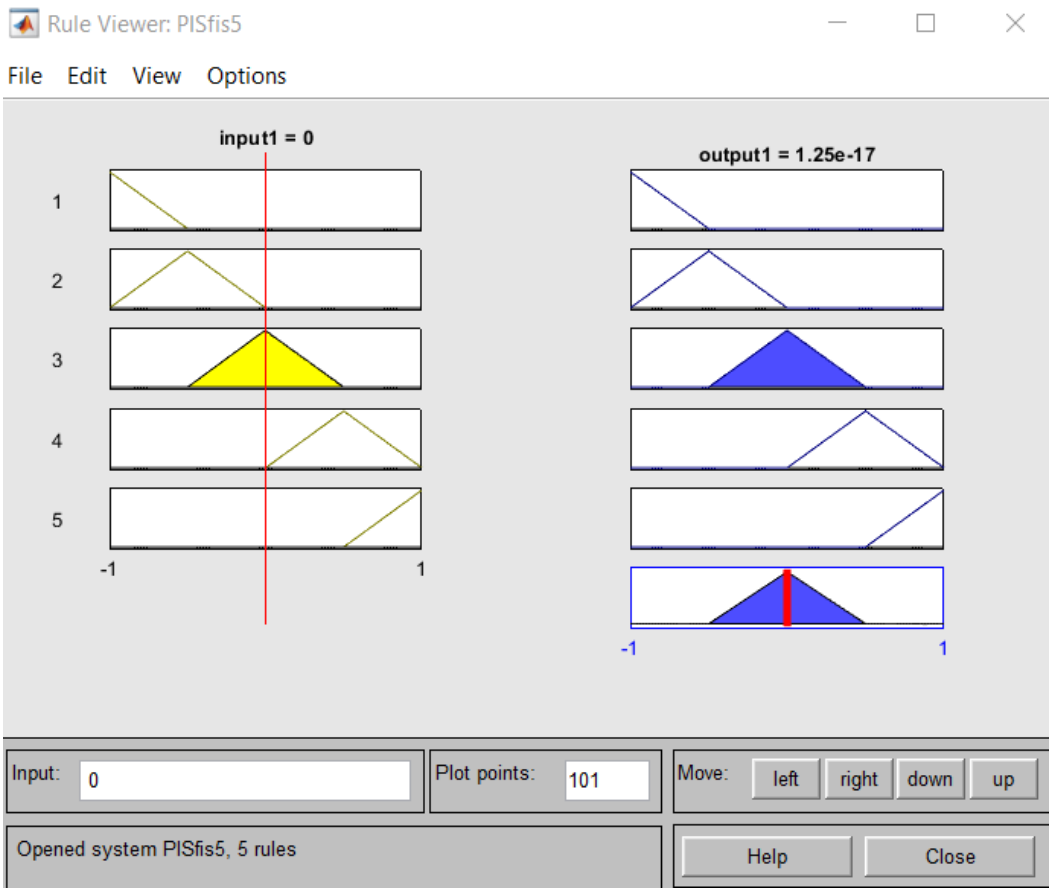


Рисунок 9.9 - Робота системи НЛР

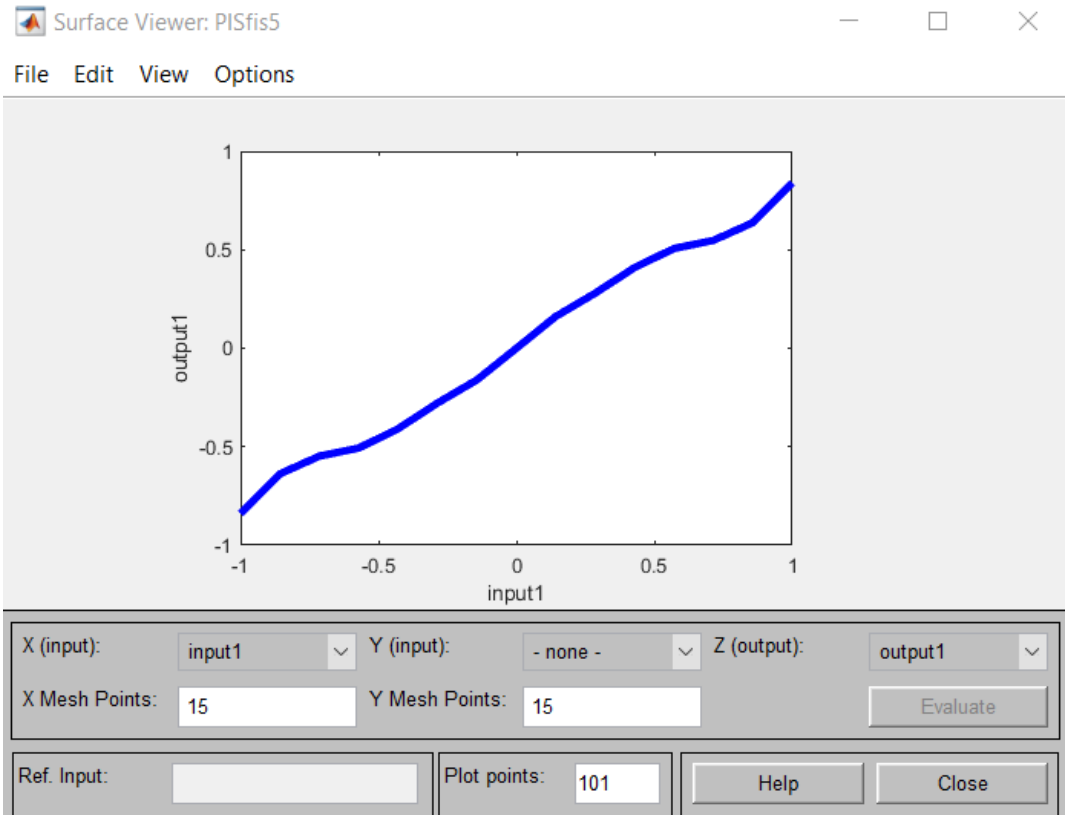


Рисунок 9.10 – Графік управління системою нечіткого висновку

Після процедур налаштування НЛР у редакторі системи нечіткого висновку здійснює запис результатів налаштування у Fuzzy Logic Controller. Для цього у вкладці File здійснюємо Export в математичну модуль нечіткого контролера From Workspase вказавши ім'я Fuzzy Logic Controller.

Після експорту закону функціонування НЛР на математичної моделі системи управління получено перехідний процес для синтезованого НЛР при різних коефіцієнтах λ (див. рис. 9.11).

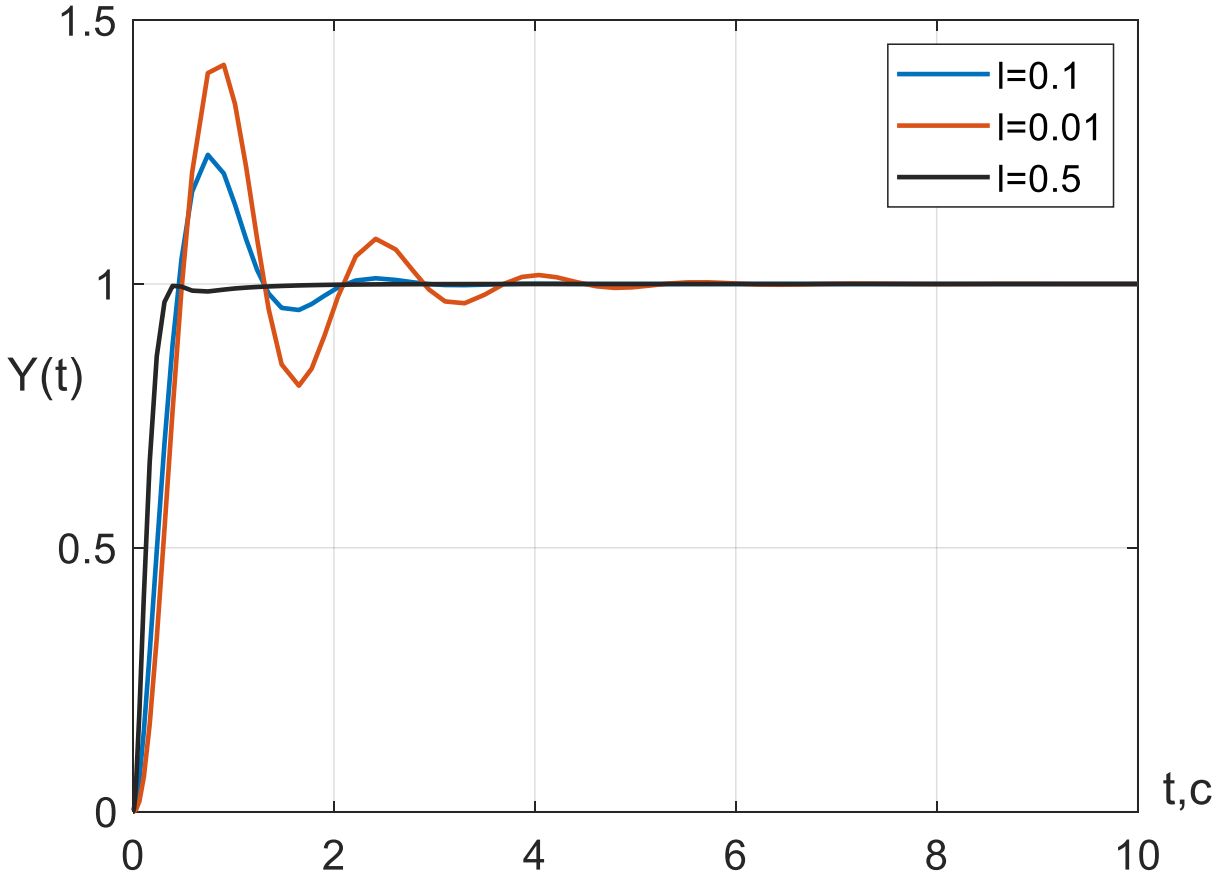


Рисунок 9.11 – Графік перехідного процесу для синтезованого НЛР при різних коефіцієнтах λ

9.2 Нечіткі супервізори

Ідея нечіткого супервізора полягає в організації дворівневої системи, в якій на нижньому рівні знаходиться звичайний ПІД-регулятор, а на верхньому рівні – НЛР (рис. 9.12).

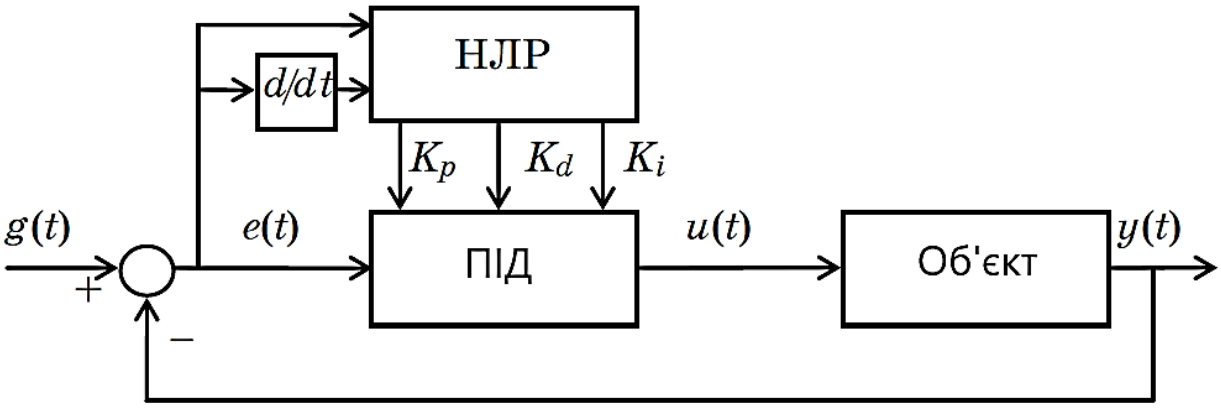


Рисунок 9.12 – Структурна схема нечіткого супервізора

Призначення НЛР полягає в тому, щоб автоматично змінювати коефіцієнти ПІД-регулятора на різних стадіях перехідного процесу. Можливі різні реалізації нечітких супервізорів, нижче розглядаються три схеми.

Перша схема безпосередньо реалізує структуру рис. 9.12. Як було показано вище, НЛР має змінний коефіцієнт посилення. У НЛР_ПІД є три вхідні змінні, тому вважатимуться, що його коефіцієнт посилення складається з трьох складових

$$K_{PID} = K_P + K_I + K_{D.}$$

Нехай помилка управління та її похідна описуються відповідно до рис. 9.13.

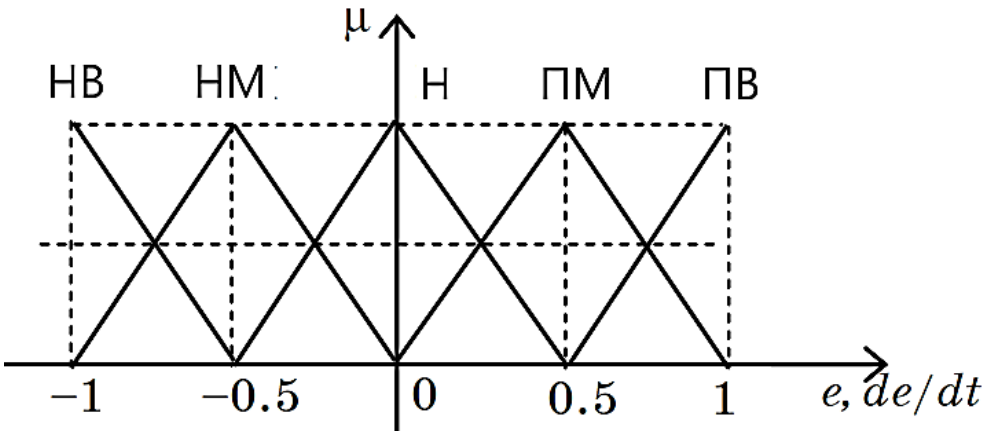


Рисунок 9.13 – Лінгвістичний опис вхідних змінних НЛР_ПІД

Лінгвістичні значення нормалізованих коефіцієнтів НЛР_ПІД можна описати за допомогою семи термів з найменуваннями «нульовий (Н)», «малий 1 (М1)», «малий 2 (М2)», «середній 1 (С1)», «середній 2 (С2)» », «Великий 1 (В1)», «Великий 2 (В2)» (рис. 9.14).

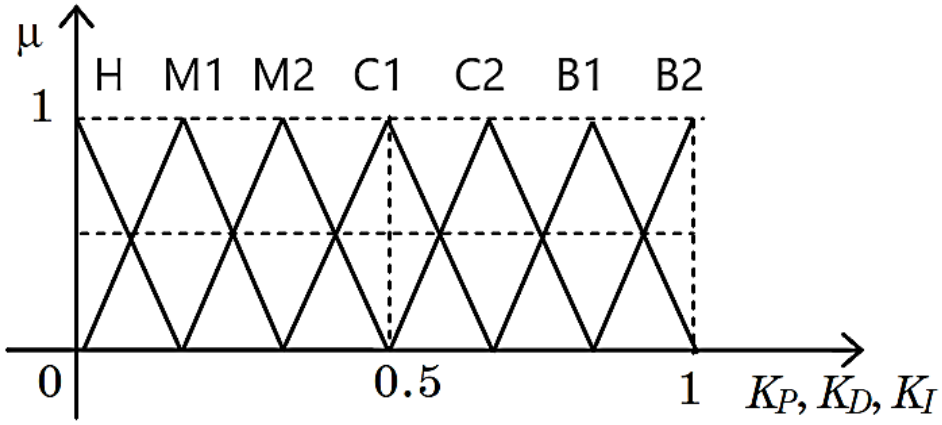


Рисунок 9.14 – Лінгвістичний опис коефіцієнтів підсилення

Для опису вимог до коефіцієнтів ПІД-регулятора розглянемо типовий графік перехідного процесу (рис. 9.15).

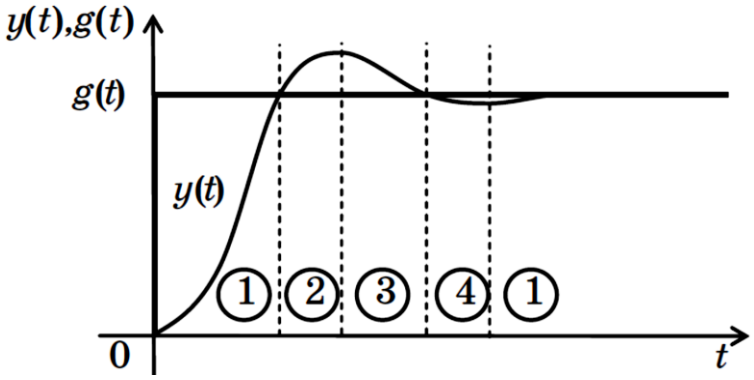


Рисунок 9.15 – Типовий перехідний процес

Розглядаючи знак помилки та її похідної, на графіку можна виділити 4 різні області:

$$1: e(t) > 0, de/dt < 0;$$
$$2: e(t) < 0, de/dt < 0;$$
$$3: e(t) < 0, de/dt > 0;$$
$$4: e(t) > 0, de/dt > 0.$$

Найбільше просто описати вимоги до коефіцієнта K_I. Він відповідає за помилку, що встановилася, і його зміну можна описати правилом: «*Якщо встановилася помилка велика, то K_I великий*». Оскільки розглядається встановлена помилка, значення похідної тут, не має значення. Однак, щоб гасити коливання в області нульової помилки, необхідно надавати K_I малі значення, коли похідна ненульова. У результаті виходить наступний опис змінного K_I (табл. 9.2).

Таблиця 9.2 – Лінгвістичні правила опису K_I

Таблиця правил		(*de*/*dt*)*				
		НВ	НМ	Н	ПМ	ПВ
Δ e*	НВ	С1	С1	С1	С1	С1
	НМ	М1	М1	М1	М1	М1
	Н	М2	М2	Н	М2	М2
	НМ	М1	М1	М1	М1	М1
	Н	С1	С1	С1	С1	С1

K_I

Коефіцієнт K_P відповідає за помилку відстеження вхідного сигналу *e*, і вимогу до його значення можна описати правилом: "*Якщо e велика, то K_P великий*". При нульовій помилці з позитивною похідної слід дещо збільшувати K_P для зменшення часу перехідного процесу. Табл. 9.3 описує ці правила.

Таблиця 9.3 – Лінгвістичні правила опису K_P

Таблиця правил		(*de*/*dt*)*				
		НВ	НМ	Н	ПМ	ПВ
Δ e*	НВ	В2	В2	В2	В2	С1
	НМ	В1	В1	В1	С2	В2
	Н	Н	Н	М2	М1	М1
	НМ	В1	В!	В1	С2	В2
	Н	В2	В2	В2	В2	В2

K_P

Коефіцієнт K_D відповідає за перерегулювання, вимогу для його значення можна описати правилом: «*Якщо перерегулювання велике, то K_D великий*». Перерегулювання виникає при наближенні *y*(*t*) до *g*(*t*), коли помилка та її похідна мають різні знаки (області 1 та 3 на рис. 9.15). У цих ситуаціях K_D повинен мати велике значення, інакше пропорційно

зменшується до нуля (див. табл. 9.4).

Таблиця 9.4 – Лінгвістичні правила опису K_D

Таблиця правил		(*de*/*dt*)*				
		НВ	НМ	Н	ПМ	ПВ
Δ e*	НВ	Н	М1	С1	С2	В2
	НМ	М1	В1	С2	В2	В2
	Н	С1	С2	С2	В2	В2
	НМ	В1	В2	В2	М1	М1
	Н	В2	В2	В2	В2	В2

K_D

Табл.9.2-9.4 можуть бути зведені в одну табл. 9.5.

Таблиця 9.5 – Правила корекції коефіцієнтів НЛР_ПІД

№	*e*	*de*/*dt*	K_p	K_D	K_I	№	*e*	*de*/*dt*	K_p	K_D	K_I
1	НВ	НВ	В2	Н	С1	14	Н	ПМ	М1	В2	М2
2	НВ	НМ	В2	М1	С1	15	Н	ПВ	М1	В2	М2
3	НВ	Н	В2	С1	С1	16	ПМ	НВ	В1	В2	М1
4	НВ	ПМ	В2	С2	С1	17	ПМ	НМ	В1	В2	М1
5	НВ	ПВ	В2	В2	С1	18	ПМ	Н	В1	В2	М1
6	НМ	НВ	В1	М1	М1	19	ПМ	ПМ	С2	В2	М1
7	НМ	НМ	В1	В1	М1	20	ПМ	ПВ	В2	В2	М1
8	НМ	Н	В1	С2	М1	21	ПВ	НВ	В2	В2	С1
9	НМ	ПМ	С2	В2	М1	22	ПВ	НМ	В2	В2	С1
10	НМ	ПВ	В2	В2	М1	23	ПВ	Н	В2	В2	С1
11	Н	НВ	Н	В1	М2	24	ПВ	ПМ	В2	В2	С1
12	Н	НМ	Н	В2	М2	25	ПВ	ПВ	В2	В2	С1
13	Н	Н	М2	В2	Н						

На рис. 9.16 показана схема моделювання НЛР_ПІД.

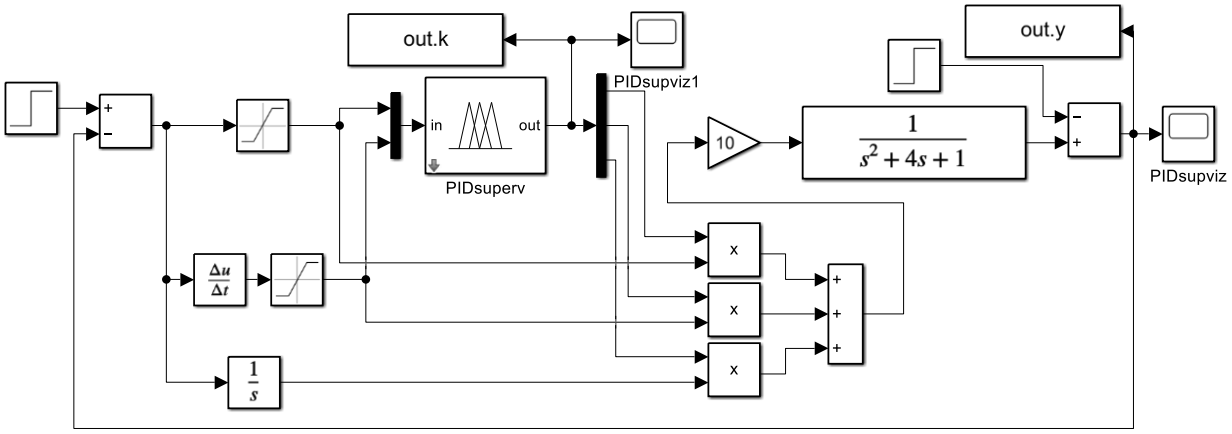


Рисунок 9.16 – Варіант математичного моделювання системи управління з нечітким супервізором ПІД регулятора

Приклад. Відповідно до рис. 9.16 сформуємо в Simulink MatLab модель системи управління *з нечітким супервізором ПІД регулятора* при x = 10.

Fuzzy Logic Controller надаємо Fis name: PIDsupv1 (ім'я може буди будь-яке).

Викликається редактор системи нечіткого висновку (Fuzzy Inference System – FIS) у MatLab командою

>> fuzzy

Дале проводиться налаштовування головного вікна редактора нечіткої логічної системи (див. рис. 9.17.).

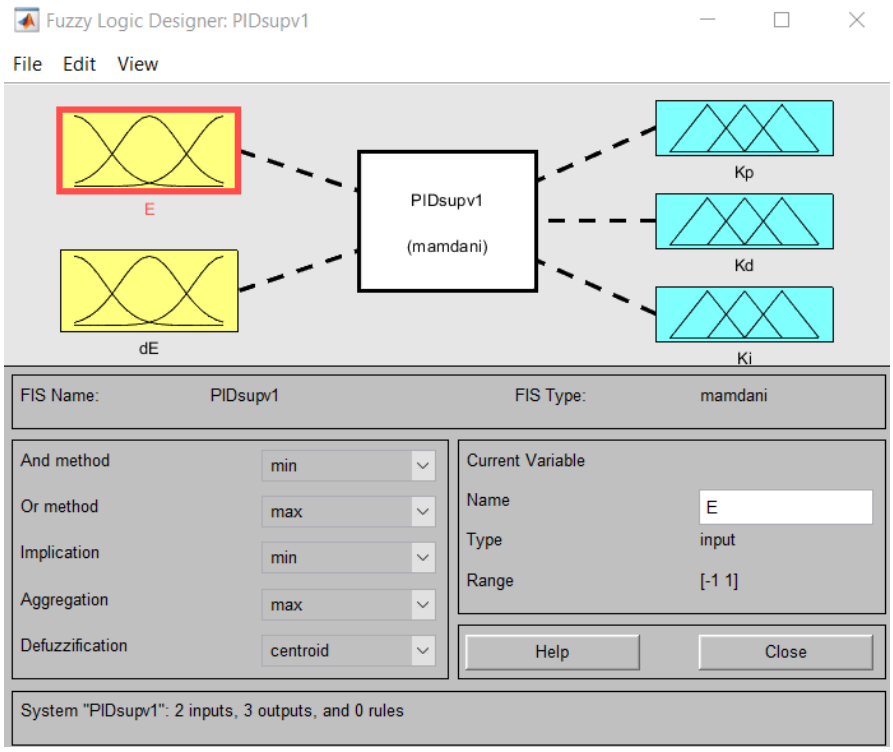


Рисунок 9.17 – Налаштування інтерфейсу FIS editor

У *FIS editor* задається опис систему нечіткого логічного висновку *Mamdani*. Для створюваної системи обирається вид логічного зв'язку (*And method – min*) та (*Or method – max*), вид імплікації (*Implication – min*), спосіб агрегування висновків правил (*Aggregation – max*) та метод дефазифікації (*Defuzzification – centroid*).

У меню *Edit* послідовно додаємо 2 вхідні змінні з розміром базової шкали (Range= [-1 1]) та 3 вихідні змінні з розміром базової шкали (*Range*= [0 1]).

Для опису вхідних логічних змінних у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної змінної трикутну функцію приналежності.

Нехай терми лінгвістичної змінної «Помилка управління» розміщуються відповідно до рис. 11.129 з 5 параметрами:

Name = "NBe"; Type = "trimf"; Param = [-1.5 -1 -0.5];
Name = "NSe"; Type = "trimf"; Param = [-1 -0.5 0];
Name = "Ze"; Type = " trimf"; Param = [-0.5 0 0.5];
Name = "PSe"; Type = "trimf"; Param = [0 0.5 1].
Name = "PBe"; Type = "trimf"; Param = [0.5 1 1.5].

Результат налаштування функцій приналежності вхідних змінних «Помилка управління» наведено на рис. 9.18.

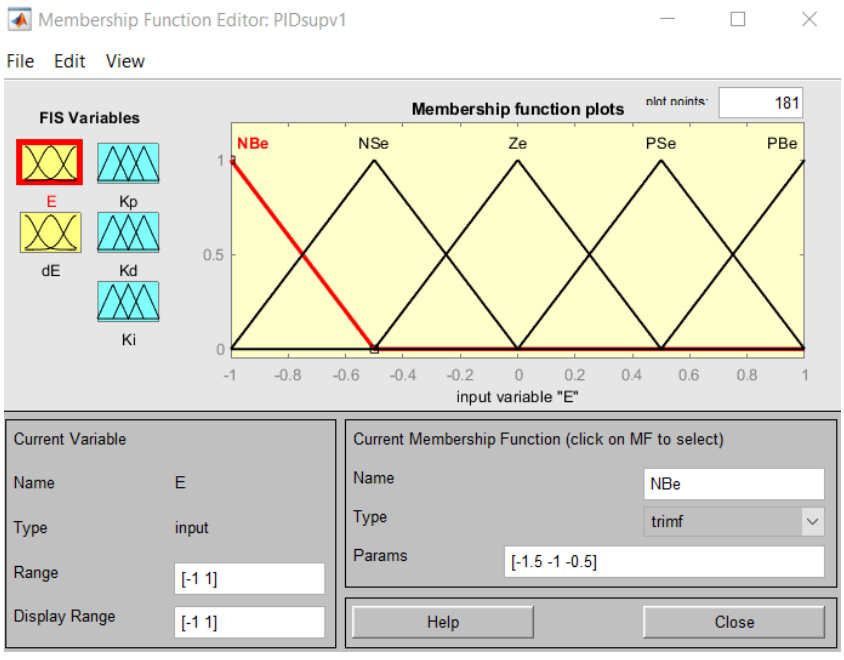


Рисунок 9.18 – Результат налаштування функцій приналежності вхідних змінних «Помилка управління»

Терми вхідної лінгвістичної змінної «Похідна помилки управління» формуємо рівномірно. Для опису вхідних логічних змінних у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної вихідної змінної трикутну функцію приналежності. З параметрами:

Name = "NBde"; Type = "trimf"; Param = [-1.5 -1 -0.5];
Name = "NSde"; Type = "trimf"; Param = [-1 -0.5 0];
Name = "Zde"; Type = " trimf"; Param = [-0.5 0 0.5];
Name = "PSde"; Type = "trimf"; Param = [0 0.5 1].
Name = "PBde"; Type = "trimf"; Param = [0.5 1 1.5].

Результат налаштування функцій приналежності вхідних змінних «Похідна помилки управління» на рис. 9.19.

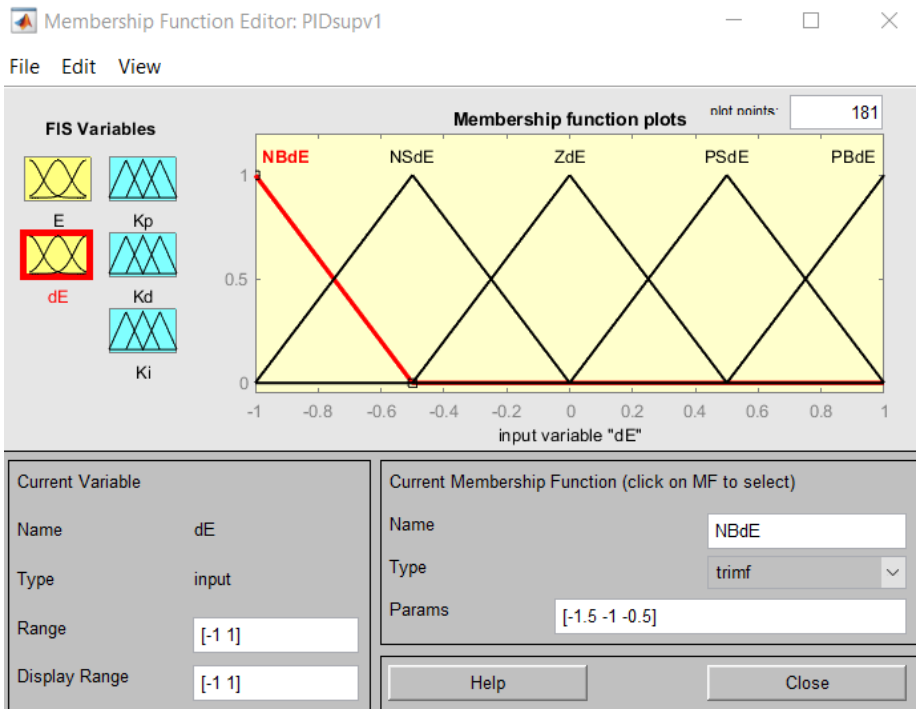


Рисунок 9.19 – Результат налаштування функцій приналежності вхідних змінних «Похідна помилки управління»

Для опису вихідних логічних змінних у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної вихідної змінної трикутну функцію приналежності.

- для K_p:

Name = "Zp"; Type = " trimf"; Param = [-0.166 0 0.166];
Name = "M1p"; Type = "trimf"; Param = [0 0.166 0.33];
Name = "M2p"; Type = " trimf"; Param = [0.166 0.33 0.5];
Name = "C1p"; Type = " trimf"; Param = [0.33 0.5 0.66];
Name = "C2p"; Type = "trimf"; Param = [0.5 0.66 0.83].
Name = "B1p"; Type = " trimf"; Param = [0.66 0.83 1];
Name = "B2p"; Type = "trimf"; Param = [0.833 1 1.166].

- для K_d:

Name = "Zd"; Type = " trimf"; Param = [-0.166 0 0.166];

Name = "M1d"; Type = "trimf"; Param = [0 0.166 0.33];
Name = "M2d"; Type = " trimf"; Param = [0.166 0.33 0.5];
Name = "C1d"; Type = " trimf"; Param = [0.33 0.5 0.66];
Name = "C2d"; Type = "trimf"; Param = [0.5 0.66 0.83].
Name = "B1d"; Type = " trimf"; Param = [0.66 0.83 1];
Name = "B2d"; Type = "trimf"; Param = [0.833 1 1.166].
- для K_I:
Name = "Zi"; Type = " trimf"; Param = [-0.166 0 0.166];
Name = "M1i"; Type = "trimf"; Param = [0 0.166 0.33];
Name = "M2i"; Type = " trimf"; Param = [0.166 0.33 0.5];
Name = "C1i"; Type = " trimf"; Param = [0.33 0.5 0.66];
Name = "C2i"; Type = "trimf"; Param = [0.5 0.66 0.83].
Name = "B1i"; Type = " trimf"; Param = [0.66 0.83 1];
Name = "B2i"; Type = "trimf"; Param = [0.833 1 1.166].

Результат налаштування функцій приналежності вхідних змінних наведено на рис. 9.20.

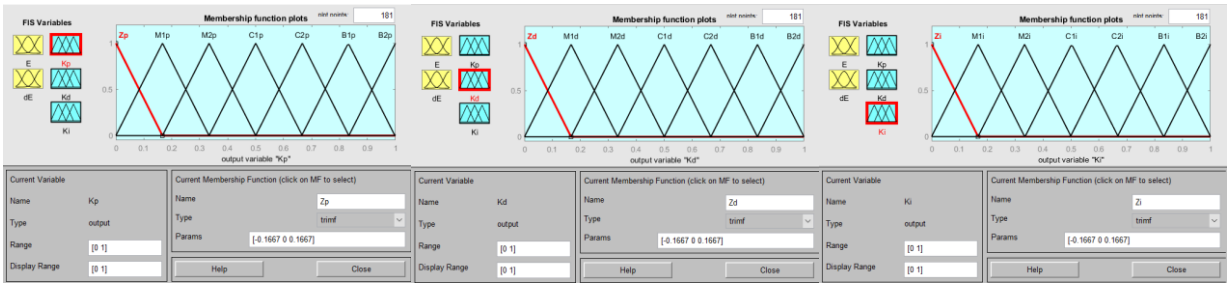


Рисунок 9.20 – Результат налаштування функцій приналежності вихідної змінної K_p, K_d, K_I

Після опису вхідні та вихідні лінгвістичних змінних, здійснюється опис правил у вікні редактора Rule Editor, кількість правил для даного випадку 25. Управляючі правила сформульовано згідно з табл. 9.5:

1. if (E is NBe) and (dE is NBdE) then (Kp is B2p)(Kd is Zd)(Ki is C1i) (1)
2. if (E is NBe) and (dE is NSdE) then (Kp is B2p)(Kd is M1d)(Ki is C1i) (1)
3. if (E is NBe) and (dE is ZdE) then (Kp is B2p)(Kd is C1d)(Ki is C1i) (1)
4. if (E is NBe) and (dE is PSdE) then (Kp is B2p)(Kd is C2d)(Ki is C1i) (1)
5. if (E is NBe) and (dE is PBdE) then (Kp is B2p)(Kd is B2d)(Ki is C1i) (1)
6. if (E is NSe) and (dE is NBdE) then (Kp is B1p)(Kd is M1d)(Ki is M1i) (1)
7. if (E is NSe) and (dE is NSdE) then (Kp is B1p)(Kd is B1d)(Ki is M1i) (1)
8. if (E is NSe) and (dE is ZdE) then (Kp is B1p)(Kd is C2d)(Ki is M1i) (1)
9. if (E is NSe) and (dE is PSdE) then (Kp is C2p)(Kd is B2d)(Ki is M1i) (1)
10. if (E is NSe) and (dE is PBdE) then (Kp is B2p)(Kd is B2d)(Ki is M1i) (1)
11. if (E is Ze) and (dE is NBdE) then (Kp is Zp)(Kd is B1d)(Ki is M2i) (1)
12. if (E is Ze) and (dE is NSdE) then (Kp is Zp)(Kd is B2d)(Ki is M2i) (1)

13. if (E is Ze) and (dE is ZdE) then (Kp is M2p)(Kd is B2d)(Ki is Zi) (1)
14. if (E is Ze) and (dE is PSdE) then (Kp is M1p)(Kd is B2d)(Ki is M2i) (1)
15. if (E is Ze) and (dE is PBdE) then (Kp is M1p)(Kd is B2d)(Ki is M2i) (1)
16. if (E is PSe) and (dE is NBdE) then (Kp is B1p)(Kd is B2d)(Ki is M1i) (1)
17. if (E is PSe) and (dE is NSdE) then (Kp is B1p)(Kd is B2d)(Ki is M1i) (1)
18. if (E is PSe) and (dE is ZdE) then (Kp is B1p)(Kd is B2d)(Ki is M1i) (1)
19. if (E is PSe) and (dE is PSdE) then (Kp is C2p)(Kd is B2d)(Ki is M1i) (1)
20. if (E is PSe) and (dE is PBdE) then (Kp is B2p)(Kd is B2d)(Ki is M1i) (1)
21. if (E is PBe) and (dE is NBdE) then (Kp is B2p)(Kd is B2d)(Ki is C1i) (1)
22. if (E is PBe) and (dE is NSdE) then (Kp is B2p)(Kd is B2d)(Ki is C1i) (1)
23. if (E is PBe) and (dE is ZdE) then (Kp is B2p)(Kd is B2d)(Ki is C1i) (1)
24. if (E is PBe) and (dE is PSdE) then (Kp is B2p)(Kd is B2d)(Ki is C1i) (1)
25. if (E is PBe) and (dE is PBdE) then (Kp is B2p)(Kd is B2d)(Ki is C1i) (1)

Налаштовування управляючих правил для даного випадку наведено на рисунку (див. рис. 9.21).

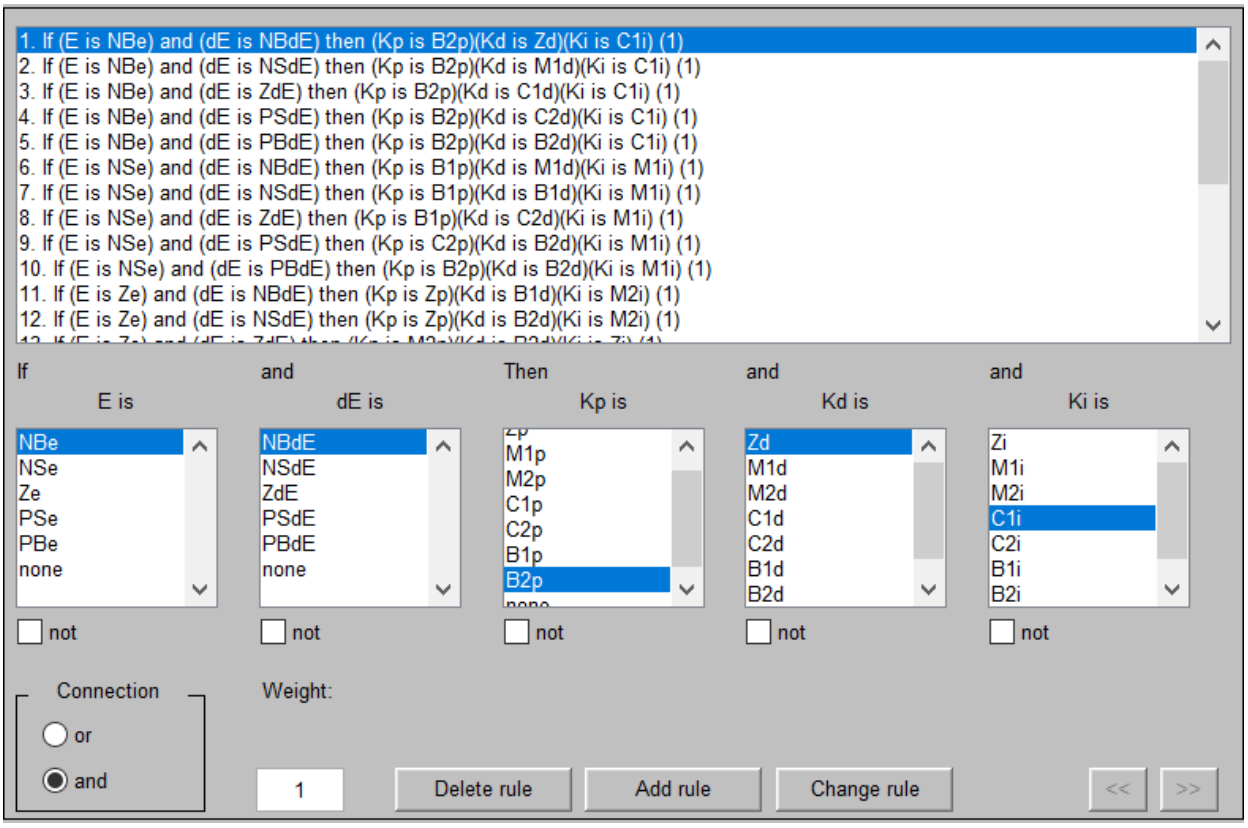


Рисунок 9.21 - Налаштовування опису правил функціонування нечіткого супервізору НЛР_ПІД

Посилки у правилах пов'язані (connection) за допомогою операції *and*. Введене правило забезпечене ваговим коефіцієнтом (*Weight*=1).

Після опису правил можливо за допомогою

- інтерфейсу *Rule Viewer* можливо переглянути роботу системи нечіткого висновку за різних вхідних даних (див. рис. 9.22).

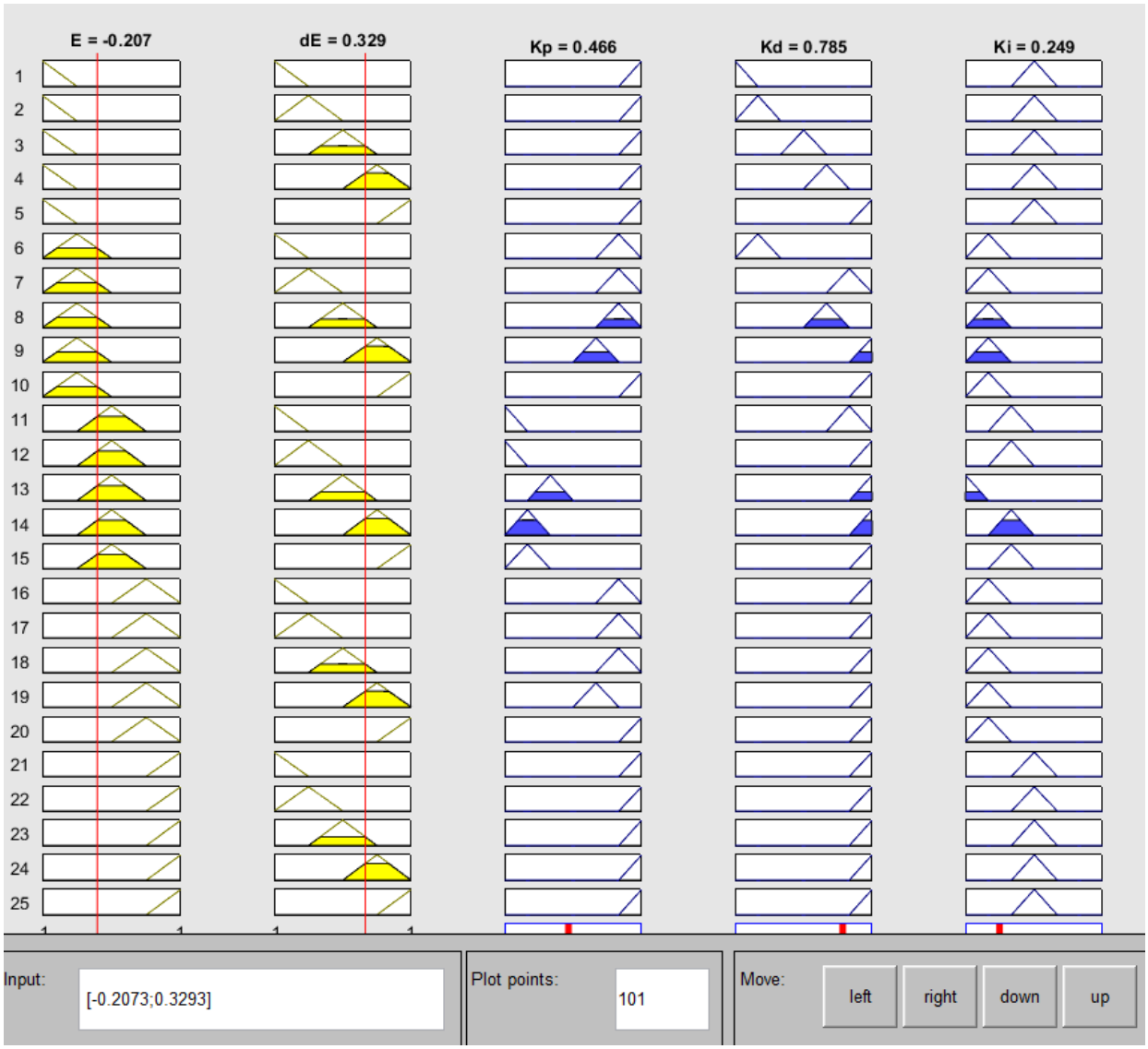


Рисунок 9.22 - Робота системи нечіткого супервізору НЛР_ПІД

− інтерфейсу Surface Viewer можливо переглянути графічне подання закону управління (рис. 9.23).

Після процедур налаштування нечіткого супервізору НЛР_ПІД у редакторі системи нечіткого висновку здійснює запис результатів налаштування у Fuzzy Logic Controller. Для цього у вкладці File здійснюємо Export в математичну модуль нечіткого контролера From Workspase вказавши ім’я Fuzzy Logic Controller.

На рис. 9.24 представлені графіки зміни коефіцієнтів ПІД-регулятора під керуванням нечіткого супервізора протягом *перехідного* процесу.

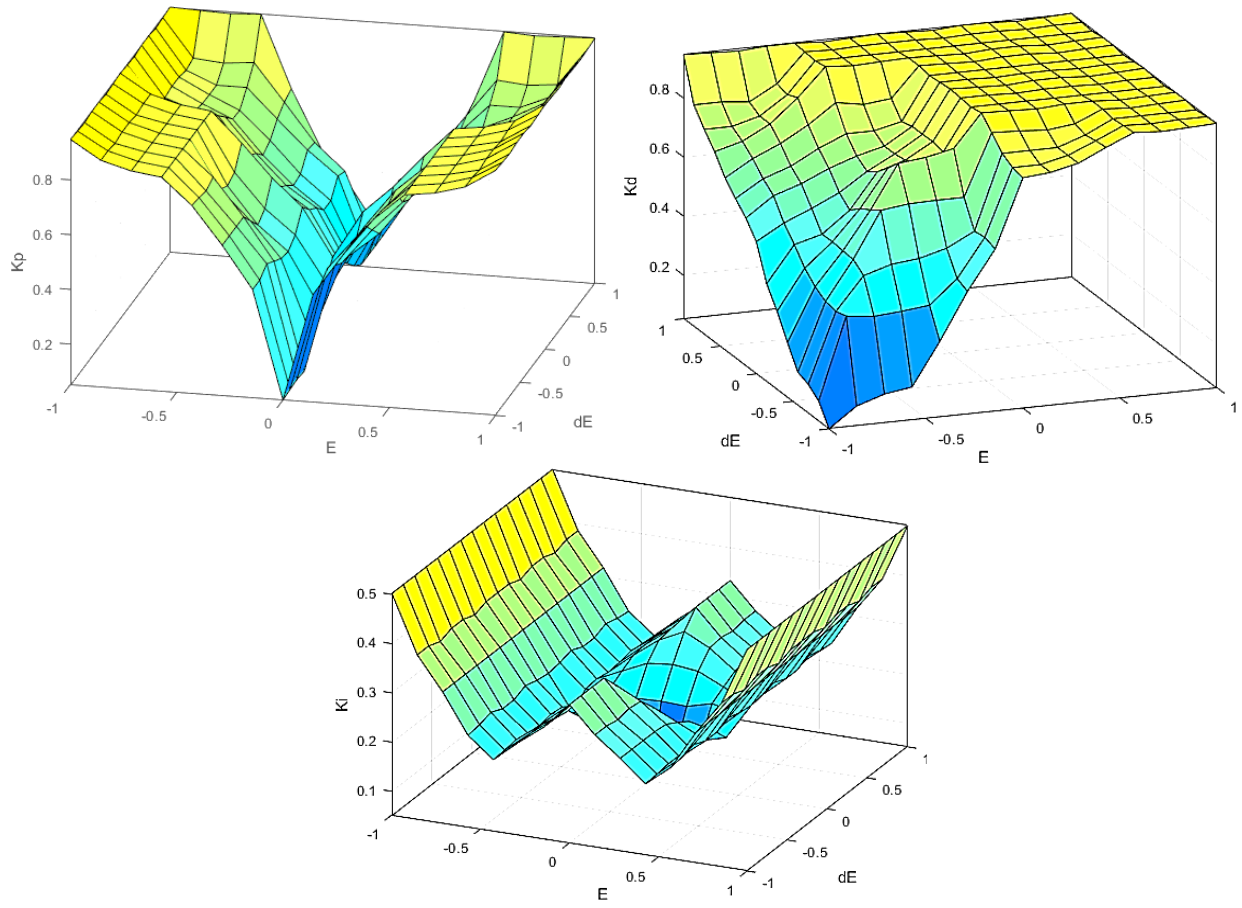


Рисунок 9.23 – Поверхні управління системою нечіткого висновку

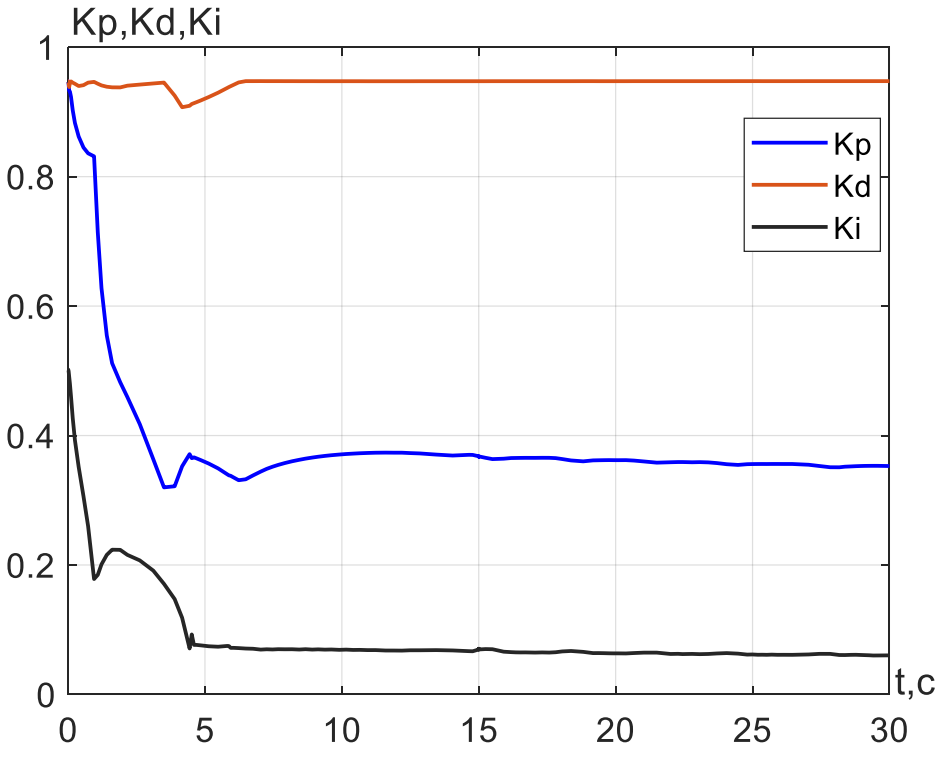


Рисунок 9.24 - Зміна коефіцієнтів ПІД-регулятора протягом перехідного процесу

На рис. 9.25 показаний перехідний процес у системі управління з нечітким супервізором.

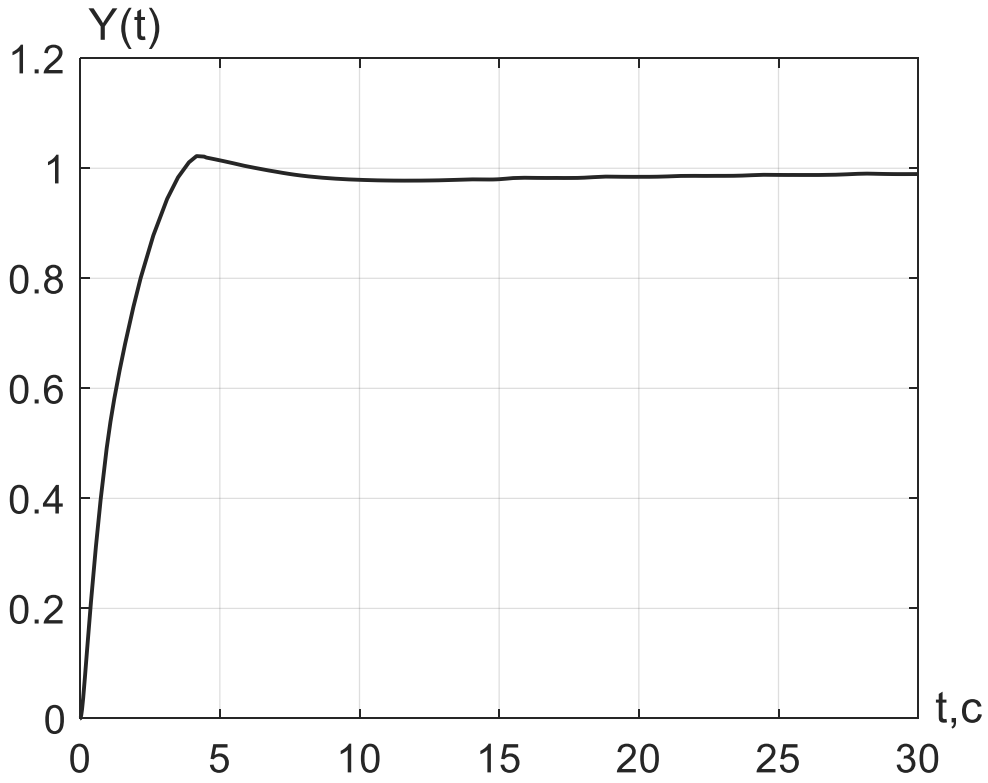


Рисунок 9.25 – Перехідний процес у системі управління з нечітким супервізором

Розглянемо другий варіант супервізорного управління, у якому відбувається інкрементальна зміна коефіцієнтів регулятора.

Нехай розглядається ПД-регулятор, у якому нечіткий супервізор може зменшувати чи збільшувати коефіцієнти посилення залежно від характеру перехідного процесу (рис. 9.26).

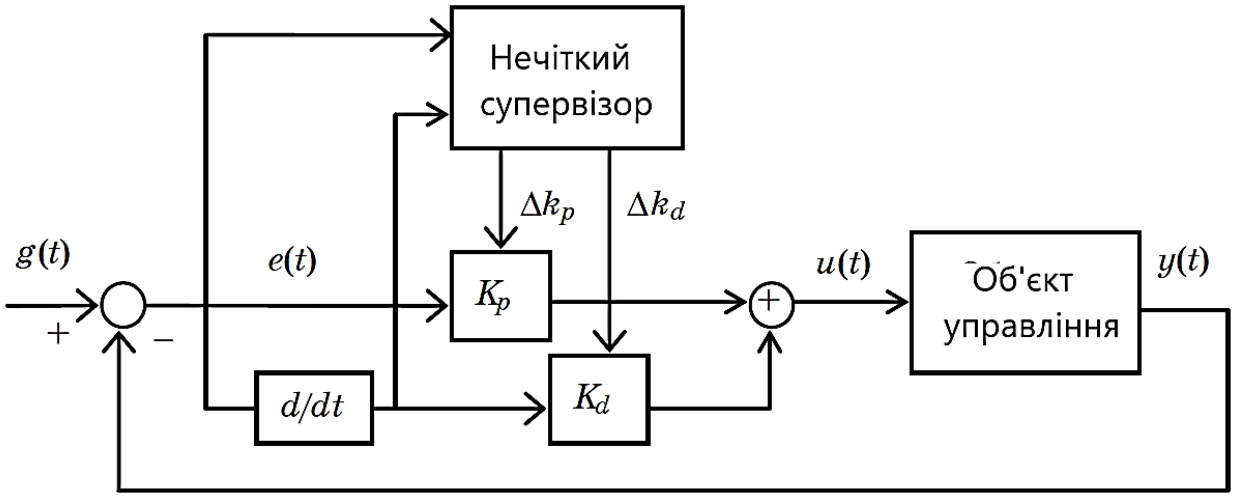


Рисунок 9.26 - Інкрементальний нечіткий супервізор

Таким чином, вихід ПД-регулятора описуватиметься формулою

$$u(t) = \left(k_p + \Delta k_p\right)e(t) + (k_d + \Delta k_d)de(t)/dt.$$

Нечітка система використовується для модифікації параметрів регулятора нижнього рівня (базового). Такий підхід може бути ефективним при керуванні нестаціонарним об'єктом, а також при неточно відомих параметрах об'єкта. Базовий регулятор може бути налаштований за стандартною методикою (наприклад, Зіглера – Ніколса), а нечіткий супервізор дозволяє варіювати коефіцієнти базового регулятора, покращуючи якість управління.

Основна ідея корекції коефіцієнтів ПД-регулятора у тому, щоб збільшувати k_p, коли керується помилка управління. Диференціальний коефіцієнт kd змінюється пропорційно k_p, тому підтримується співвідношення k_d/k_p, отримане при початковому налаштуванні регулятора. Такий закон корекції забезпечується формулами:

$$\Delta k_p = y(t)k_p;$$
$$\Delta k_d = y(t)k_d.$$

де $y(t)$ - сигнал, що виробляється нечітким супервізором.

Закон роботи нечіткого супервізора відрізняється від таблиці логічних змінних звичайного НЛР, його можна обґрунтувати, розглядаючи типовий графік перехідного процесу (рис. 9.15), який на фазовій площині набуває наступного вигляду (рис. 9.27).

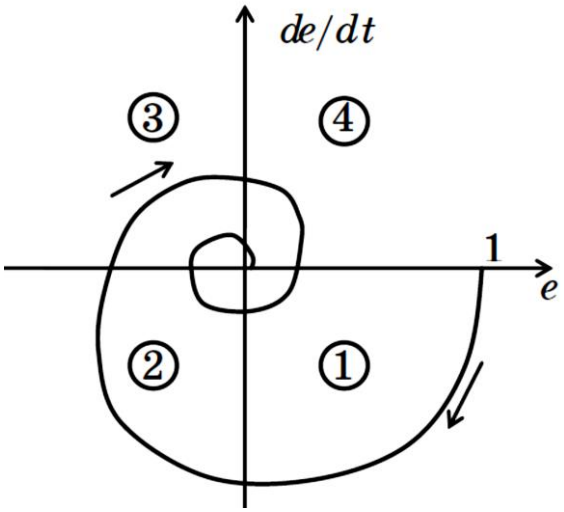


Рисунок 9.27 – Перехідний процес на фазовій площині

На початку перехідного процесу $e(t)$ велика позитивна та de/dt нульова. У регуляторі нижнього рівня використовуються номінальні параметри.

Потім $e(t)$ починає поступово зменшуватися, а de/dt зростає у негативному напрямку (зона 1). Вихід супервізора поступово збільшується від нуля до максимального значення, щоб зменшити час наростання.

У зоні 2 $e(t)$ негативна, і система повинна повільно рухатися вниз зменшення перерегулювання. Це досягається зменшенням контрольованого параметра, і вихід супервізора у цій галузі негативний.

Як тільки перерегулювання досягло пікового значення, знак de/dt стає позитивним (зона 3), і вихід супервізора повинен збільшувати своє значення, доки система не потрапить до регіону 4.

У зоні 4 $e(t)$ знову позитивна, і вихідний сигнал об'єкта наближається до стану, що встановився. Тут потрібно поступово зменшувати посилення, щоб обмежити перерегулювання.

Якщо $e(t)$ нульова, вихід супервізора протилежний за знаком значення de/dt.

Якщо de/dt нульова, то вихід супервізора збігається за знаком з $e(t)$, і має бути більшим у області малих $e(t)$ зменшення статичної помилки.

Зроблені зауваження дозволяють описати поведінку нечіткого супервізора за допомогою таблиці лінгвістичних змінних, що показана на рис. 9.28.

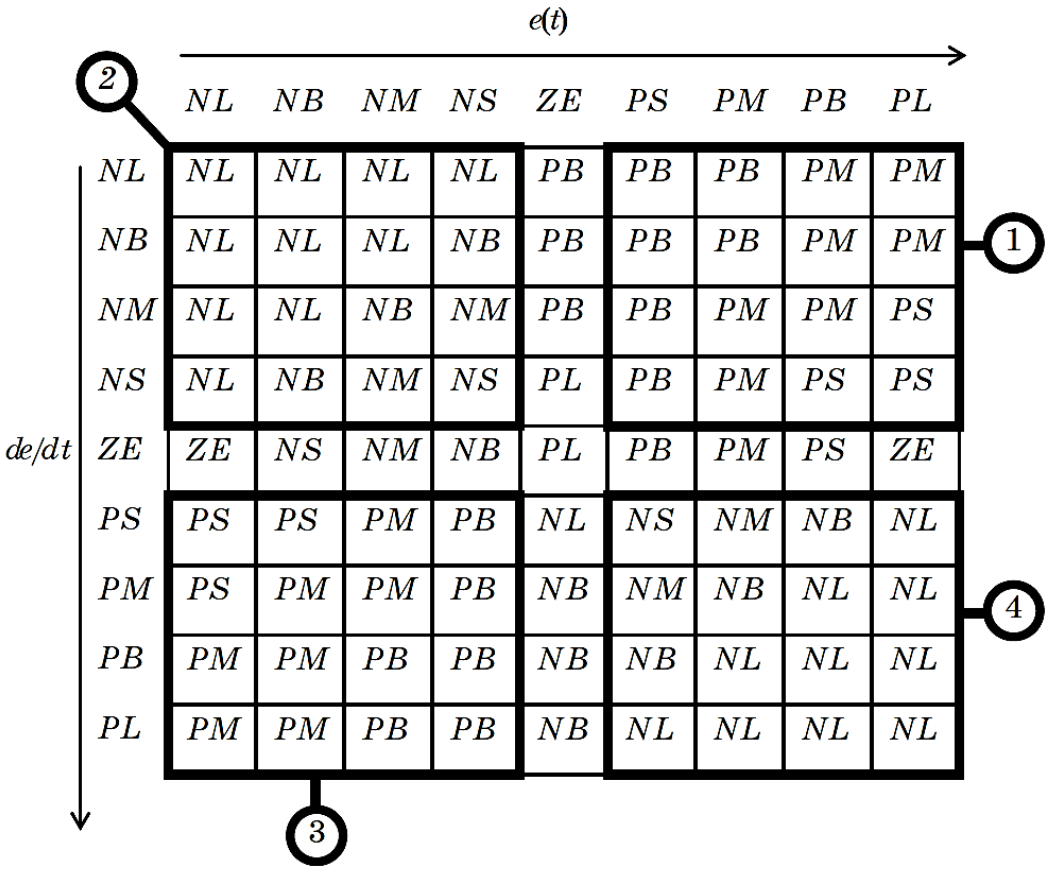


Рисунок 9.28 – Таблиця лінгвістичних правил нечіткого супервізора (цифрами позначені регіони фазової площини)

При опису передбачається використання одноманітного лінгвістичного опису нормалізованих $e(t)$, de/dt та $y(t)$ у вигляді, показаному на рис. 9.29, де використані такі скорочення: NL – negative large (негативне дуже велике), NB – negative big (негативне велике), NM – negative medium (негативне середнє), NS – negative small (негативне мале), ZE – zero (нульове) PS – positive small (позитивне мале), PM – positive medium (позитивне середнє), PB – positive big (позитивне велике), PL – positive large (позитивне дуже велике).

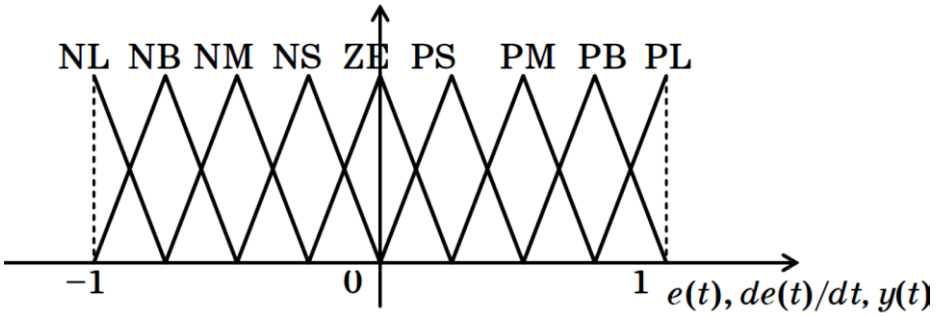


Рисунок 9.29 – Лінгвістичний опис помилки та її похідної

Розглянемо приклад використання нечіткого супервізора. Нехай є система управління з ПД-регулятором, коефіцієнти якого розраховані за методом Зіглера – Ніколса (див. рис. 9.30).

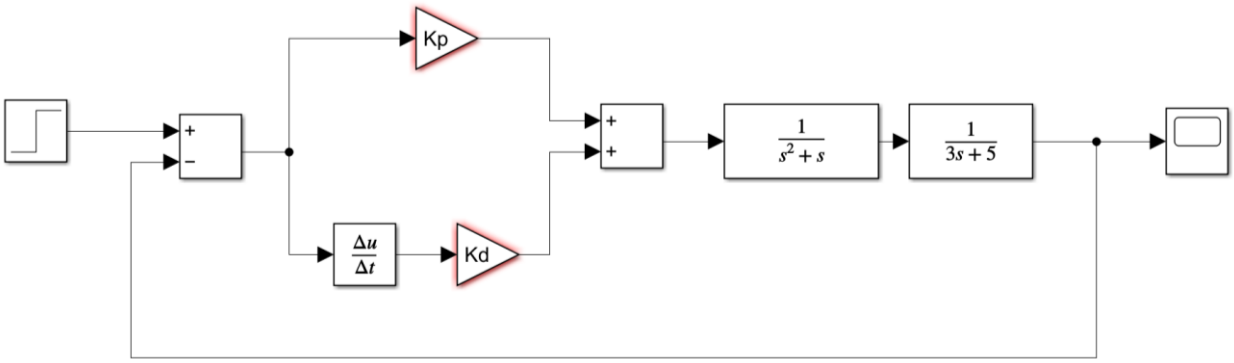


Рисунок 9.30 – Структурна схема математичної моделі системи управління з ПД-регулятором у MatLab Simulink

Налаштування регулятора методом Зіглера – Ніколса полягає в тому, щоб обчислити або підібрати оптимальні значення коефіцієнтів:

- K_P - коефіцієнт посилення пропорційної складової;
- K_I - коефіцієнт посилення інтегруючої складової;
- K_D - коефіцієнт посилення диференціюючої складової.

За великим рахунком, при використанні ПД, ПІ, ПІД-регулятора необхідно побудувати модель усієї системи в цілому та математично обчислити необхідні значення коефіцієнтів. У такому разі значення

можна розрахувати точно. Але на практиці математичний розрахунок коефіцієнтів - завдання далеко не тривіальне і вимагає глибоких знань теорії автоматичного управління, тому в більшості випадків використовуються інші, спрощені методи налаштування.

Метод полягає у послідовному виконанні наступних операцій:

- коефіцієнти регулятора дорівнюємо 0.
- встановлюємо певне цільове значення регульованого (вихідного) параметра.
- поступово починаємо збільшувати пропорційний коефіцієнт та стежимо за реакцією системи.
- за певного значення $K=K_P$ виникнуть незагасні коливання регульованої величини.
- фіксуємо це значення, а також період коливань системи.

У цьому практична частина методу закінчується. З отриманих значень розраховуємо коефіцієнти:

$$K_P = 0.6K;$$
$$K_I = \frac{2K_P}{T};$$
$$K_D = \frac{K_P T}{8}.$$

де K - коефіцієнт пропорційної складової, у якому виникли коливання, T – період цих коливань.

Налаштуємо коефіцієнти ПД регулятора методом Зиглера-Нікольса для даного випадку.

1. Задаємо на математичної моделі K_D =0, а коефіцієнт пропорційної складової регулятора збільшуємо до значення при якому у системі виникає коливальний перехідний процес з однаковим періодом коливань. У даному випадку такі коливання у системі управління настають при значенні $K=K_P$=13. (див. рис. 9.31).

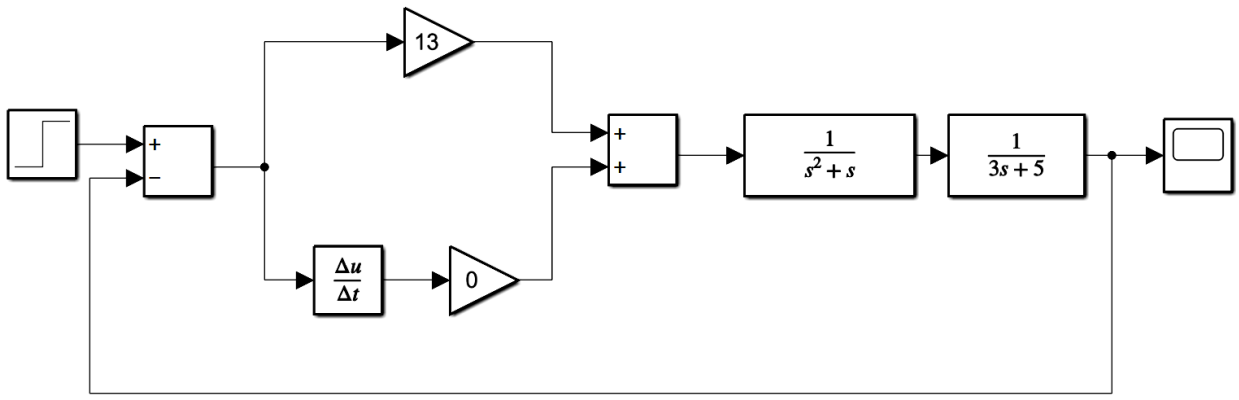


Рисунок 9.31 – Структурна схема налаштування регулятора математичної моделі системи управління у MatLab Simulink

2. На графіку перехідного процесу визначається період коливань *T*=4,9 с (див. рис. 9.32).

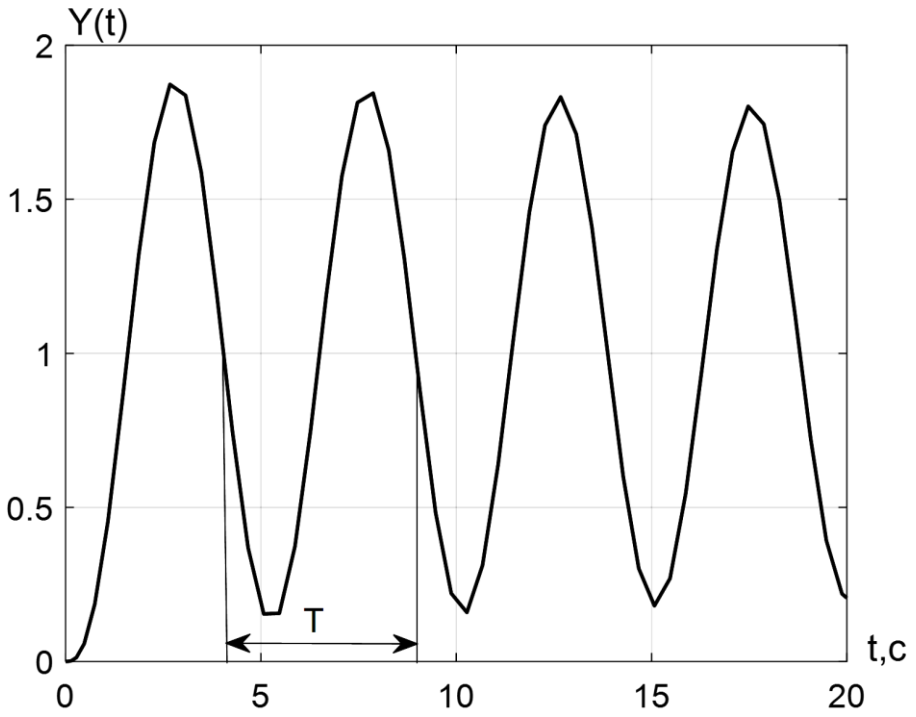


Рисунок 9.32 – Графік перехідного процесу системи управління при К=К$_P$=13

3. Розраховуємо коефіцієнти передачі ПД-регулятора

$$K_P = 0.6K = 0{,}6 \cdot 13 = 7{,}8;$$
$$K_D = \frac{K_P T}{8} = \frac{7{,}8 \cdot 4{,}9}{8} = 4{,}8.$$

Заносимо результати розрахунку коефіцієнтів ПД-регулятору до математичної моделі системи (див. рис 9.33.)

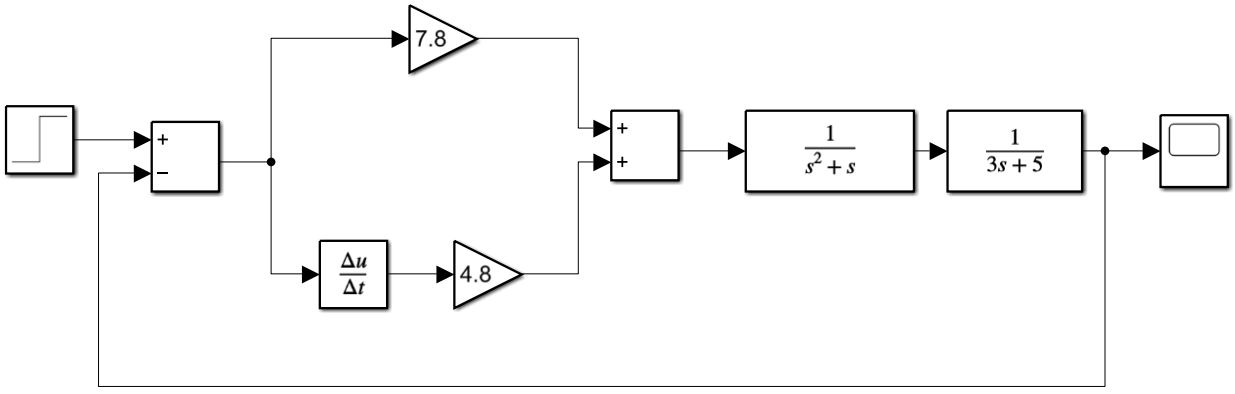


Рисунок 9.33 - Структурна схема математичної моделі системи управління з ПД-регулятором

Перехідний процес для системи управління наведеної на рисунку 9.33 має вигляд який зображено на рис. 9.34.

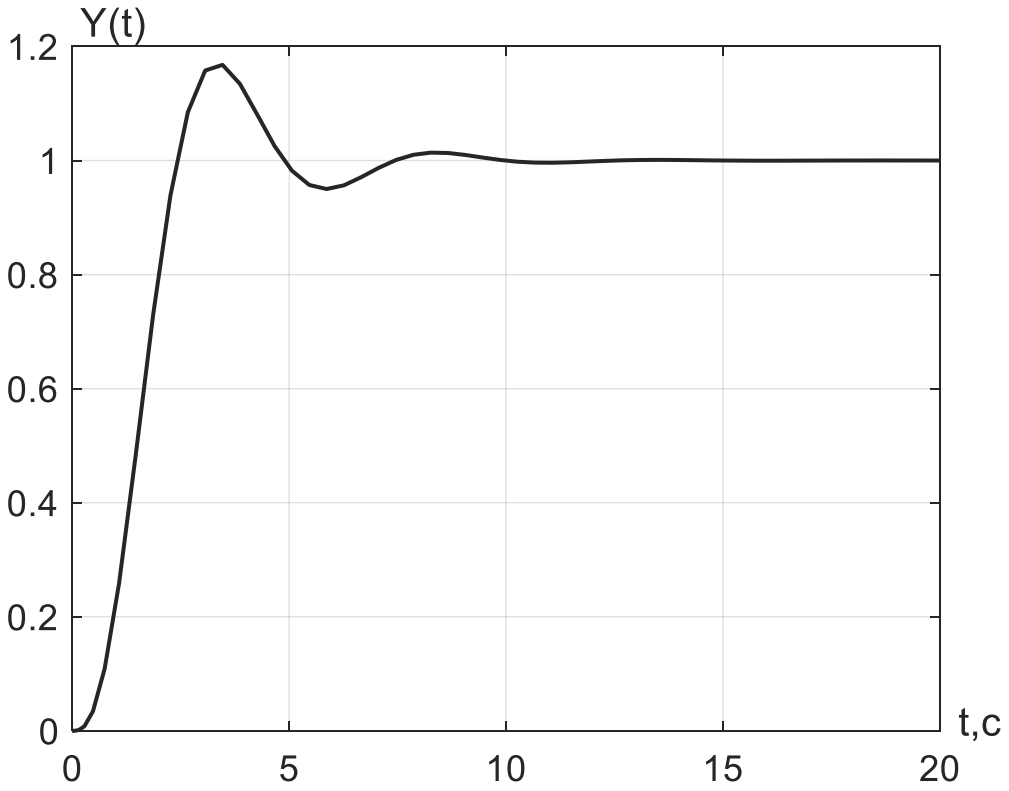


Рисунок 9.34 – Перехідний процес с системі управління з ПД-регулятором

Приклад. Проведемо синтез ПД-регулятора з нечітким супервізором якому відбувається інкрементальна зміна коефіцієнтів регулятора нечітким супервізором ПД регулятора. Сформуємо в Simulink MatLab модель системи управління (див. рис. 9.35).

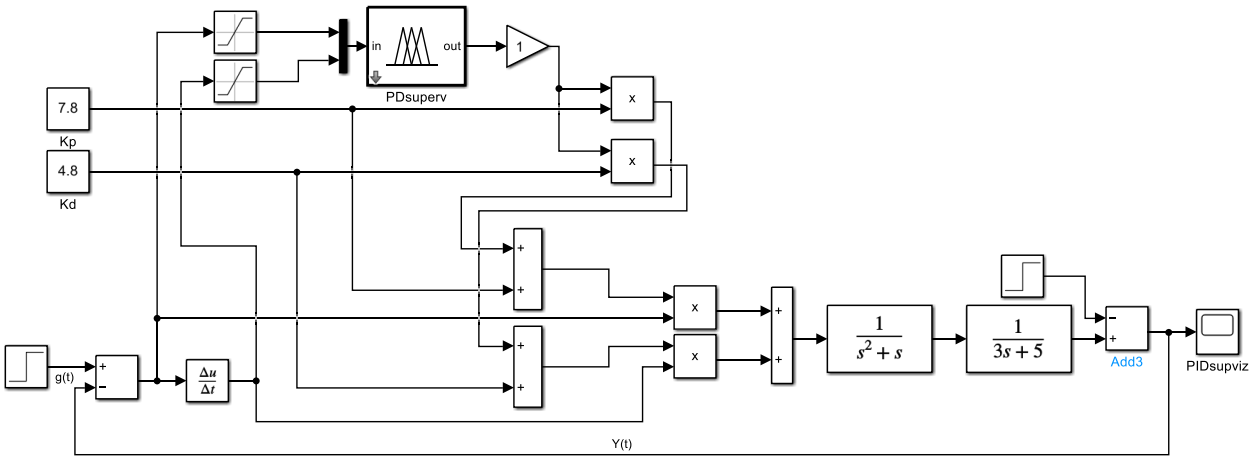


Рисунок 9.35 – Структурна схема математичної моделі системи управління з нечітким супервізором інкрементального типу

Fuzzy Logic Controller надаємо Fis name: PDsupv2 (ім'я може буди будь-яке).

Викликається редактор системи нечіткого висновку (Fuzzy Inference System – FIS) у MatLab командою

>> fuzzy

Дале проводиться налаштовування головного вікна редактора нечіткої логічної системи (див. рис. 9.36).

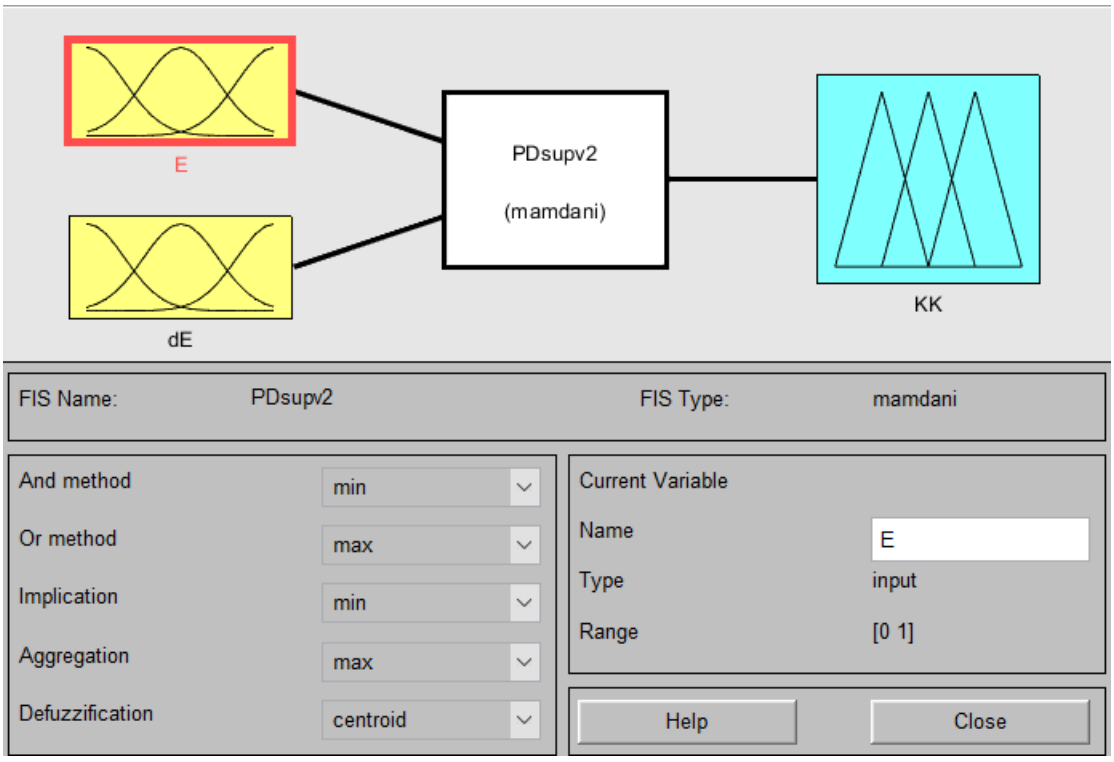


Рисунок 9.36 – Налаштування інтерфейсу FIS editor

У *FIS editor* задається опис систему нечіткого логічного висновку *Mamdani*. Для створюваної системи обирається вид логічного зв'язку (*And method – min*) та (*Or method – max*), вид імплікації (*Implication – min*), спосіб агрегування висновків правил (*Aggregation – max*) та метод дефазифікації (*Defuzzification – centroid*).

У меню *Edit* послідовно додаємо 2 вхідні змінні з розміром базової шкали (Range= [-1 1]) та вихідну змінну з розміром базової шкали (*Range*= [0 1]).

Для опису вхідних логічних змінних у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної змінної трикутну функцію приналежності.

Нехай терми лінгвістичної змінної «Помилка управління» розміщуються відповідно до рис. 9.37 з 9 параметрами функцій приналежності:

Name = "NLe"; Type = "trimf"; Param = [-1.125 -1 -0.75];

Name = "NBe"; Type = "trimf"; Param = [-1 -0.75 -0.5];
Name = "NMe"; Type = "trimf"; Param = [-0.75 -0.5 -0.25];
Name = "NSe"; Type = "trimf"; Param = [-0.5 -0.25 0];
Name = "Ze"; Type = " trimf"; Param = [-0.25 0 0.25];
Name = "PSe"; Type = "trimf"; Param = [0 0.25 0.5].
Name = "PMe"; Type = "trimf"; Param = [0.25 0.5 0.75].
Name = "PBe"; Type = "trimf"; Param = [0.5 0.75 1].
Name = "PLe"; Type = "trimf"; Param = [0.75 1 1.25].

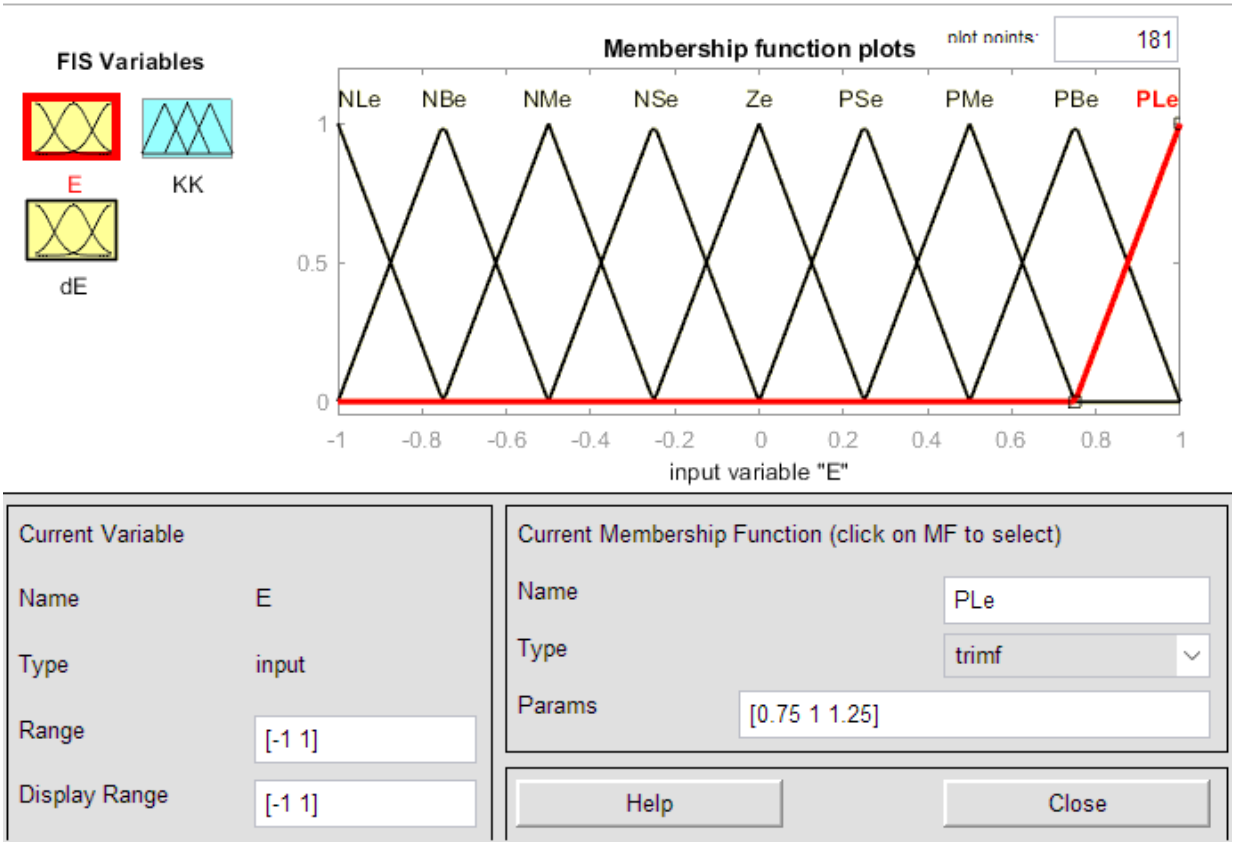


Рисунок 9.37 – Результат налаштування функцій приналежності вхідних змінних «Помилка управління»

Терми вхідної лінгвістичної змінної «Похідна помилки управління» формуємо рівномірно. Для опису вхідних логічних змінних у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної вихідної змінної трикутну функцію приналежності. З параметрами:

Name = "NLdE"; Type = "trimf"; Param = [-1.125 -1 -0.75];
Name = "NBdE"; Type = "trimf"; Param = [-1 -0.75 -0.5];
Name = "NMdE"; Type = "trimf"; Param = [-0.75 -0.5 -0.25];
Name = "NSdE"; Type = "trimf"; Param = [-0.5 -0.25 0];
Name = "ZdE"; Type = " trimf"; Param = [-0.25 0 0.25];
Name = "PSdE"; Type = "trimf"; Param = [0 0.25 0.5].

Name = "PMdE"; Type = "trimf"; Param = [0.25 0.5 0.75].
Name = "PBdE"; Type = "trimf"; Param = [0.5 0.75 1].
Name = "PLdE"; Type = "trimf"; Param = [0.75 1 1.25].

Результат налаштування функцій приналежності вхідних змінних «Похідна помилки управління» на рис. 9.38.

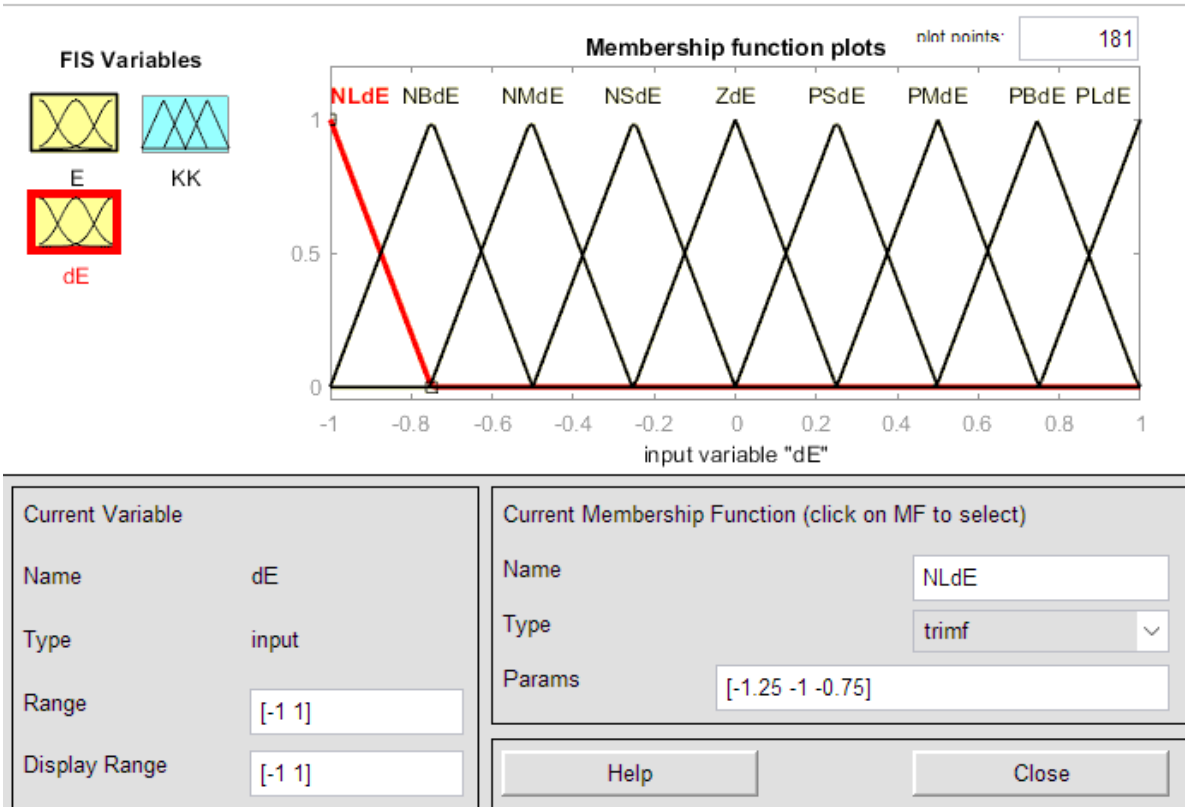


Рисунок 9.38 – Результат налаштування функцій приналежності вхідних змінних «Похідна помилки управління»

Для опису вихідної логічної змінної у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної вихідної змінної трикутну функцію приналежності.

\
Name = "NL"; Type = "trimf"; Param = [-1.125 -1 -0.75];
Name = "NB"; Type = "trimf"; Param = [-1 -0.75 -0.5];
Name = "NM"; Type = "trimf"; Param = [-0.75 -0.5 -0.25];
Name = "NS"; Type = "trimf"; Param = [-0.5 -0.25 0];
Name = "Z"; Type = " trimf"; Param = [-0.25 0 0.25];
Name = "PS"; Type = "trimf"; Param = [0 0.25 0.5].
Name = "PM"; Type = "trimf"; Param = [0.25 0.5 0.75].
Name = "PB"; Type = "trimf"; Param = [0.5 0.75 1].
Name = "PL"; Type = "trimf"; Param = [0.75 1 1.25].

Результат налаштування функцій приналежності вхідних змінних наведено на рис. 9.39.

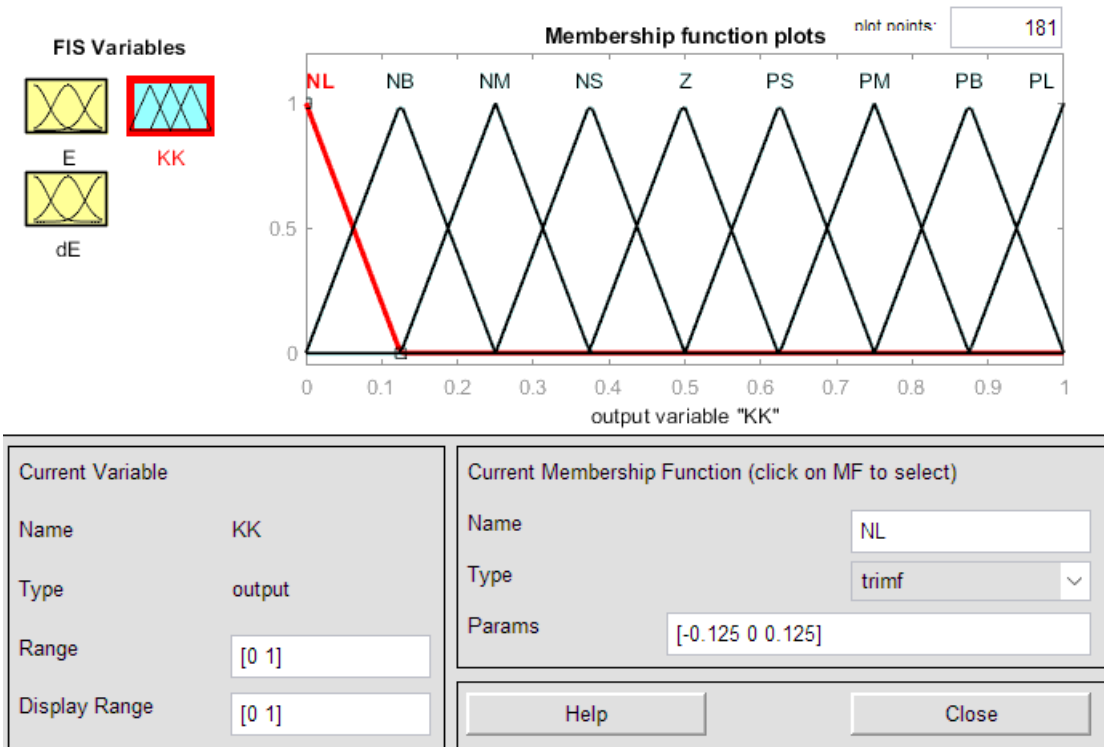


Рисунок 9.39 – Результат налаштування функцій приналежності вихідної змінної

Після опису вхідні та вихідні лінгвістичних змінних, здійснюється опис правил у вікні редактора Rule Editor, кількість правил для даного випадку 25. Управляючі правила сформульовано згідно з рис. 9.28:

1. if (E is NLe) and (dE is NLdE) then (KK is NL) (1)
2. if (E is NLe) and (dE is NBdE) then (KK is NL) (1)
3. if (E is NLe) and (dE is NMdE) then (KK is NL) (1)
4. if (E is NLe) and (dE is NSdE) then (KK is NL) (1)
5. if (E is NLe) and (dE is ZdE) then (KK is PB) (1)
6. if (E is NLe) and (dE is PSdE) then (KK is PB) (1)
7. if (E is NLe) and (dE is PMdE) then (KK is PB) (1)
8. if (E is NLe) and (dE is PBdE) then (KK is PM) (1)
9. if (E is NLe) and (dE is PLE) then (KK is PM) (1)
10. if (E is NBe) and (dE is NLdE) then (KK is NL) (1)
11. if (E is NBe) and (dE is NBdE) then (KK is NL) (1)
12. if (E is NBe) and (dE is NMdE) then (KK is NL) (1)
13. if (E is NBe) and (dE is NSdE) then (KK is NB) (1)
14. if (E is NBe) and (dE is ZdE) then (KK is PB) (1)
15. if (E is NBe) and (dE is PSdE) then (KK is PB) (1)
16. if (E is NBe) and (dE is PMdE) then (KK is PB) (1)
17. if (E is NBe) and (dE is PBdE) then (KK is PM) (1)
18. if (E is NBe) and (dE is PLE) then (KK is PM) (1)
19. if (E is NMe) and (dE is NLdE) then (KK is NL) (1)
20. if (E is NMe) and (dE is NBdE) then (KK is NL) (1)

21. if (E is NMe) and (dE is NMdE) then (KK is NB) (1)
22. if (E is NMe) and (dE is NSdE) then (KK is NM) (1)
23. if (E is NMe) and (dE is ZdE) then (KK is PB) (1)
24. if (E is NMe) and (dE is PSdE) then (KK is PB) (1)
25. if (E is NMe) and (dE is PMdE) then (KK is PM) (1)
26. if (E is NMe) and (dE is PBdE) then (KK is PM) (1)
27. if (E is NMe) and (dE is PLE) then (KK is PS) (1)
28. if (E is NSe) and (dE is NLdE) then (KK is NL) (1)
29. if (E is NSe) and (dE is NBdE) then (KK is NB) (1)
30. if (E is NSe) and (dE is NMdE) then (KK is NM) (1)
31. if (E is NSe) and (dE is NSdE) then (KK is NS) (1)
32. if (E is NSe) and (dE is ZdE) then (KK is PL) (1)
33. if (E is NSe) and (dE is PSdE) then (KK is PB) (1)
34. if (E is NSe) and (dE is PMdE) then (KK is PM) (1)
35. if (E is NSe) and (dE is PBdE) then (KK is PS) (1)
36. if (E is NSe) and (dE is PLE) then (KK is PS) (1)
37. if (E is Ze) and (dE is NLdE) then (KK is Z) (1)
38. if (E is Ze) and (dE is NBdE) then (KK is NS) (1)
39. if (E is Ze) and (dE is NMdE) then (KK is NM) (1)
40. if (E is Ze) and (dE is NSdE) then (KK is NB) (1)
41. if (E is Ze) and (dE is ZdE) then (KK is PL) (1)
42. if (E is Ze) and (dE is PSdE) then (KK is PB) (1)
43. if (E is Ze) and (dE is PMdE) then (KK is PM) (1)
44. if (E is Ze) and (dE is PBdE) then (KK is PS) (1)
45. if (E is Ze) and (dE is PLE) then (KK is Z) (1)
46. if (E is PSe) and (dE is NLdE) then (KK is PS) (1)
47. if (E is PSe) and (dE is NBdE) then (KK is PS) (1)
48. if (E is PSe) and (dE is NMdE) then (KK is PM) (1)
49. if (E is PSe) and (dE is NSdE) then (KK is PB) (1)
50. if (E is PSe) and (dE is ZdE) then (KK is NL) (1)
51. if (E is PSe) and (dE is PSdE) then (KK is NS) (1)
52. if (E is PSe) and (dE is PMdE) then (KK is NM) (1)
53. if (E is PSe) and (dE is PBdE) then (KK is NB) (1)
54. if (E is PSe) and (dE is PLE) then (KK is NL) (1)
55. if (E is PMe) and (dE is NLdE) then (KK is PS) (1)
56. if (E is PMe) and (dE is NBdE) then (KK is PM) (1)
57. if (E is PMe) and (dE is NMdE) then (KK is PM) (1)
58. if (E is PMe) and (dE is NSdE) then (KK is PB) (1)
59. if (E is PMe) and (dE is ZdE) then (KK is NB) (1)
60. if (E is PMe) and (dE is PSdE) then (KK is NM) (1)
61. if (E is PMe) and (dE is PMdE) then (KK is NB) (1)
62. if (E is PMe) and (dE is PBdE) then (KK is NL) (1)
63. if (E is PMe) and (dE is PLE) then (KK is NL) (1)
64. if (E is PBe) and (dE is NLdE) then (KK is PM) (1)
65. if (E is PBe) and (dE is NBdE) then (KK is PM) (1)

66. if (E is PBe) and (dE is NMdE) then (KK is PB) (1)
67. if (E is PBe) and (dE is NSdE) then (KK is PB) (1)
68. if (E is PBe) and (dE is ZdE) then (KK is NB) (1)
69. if (E is PBe) and (dE is PSdE) then (KK is NB) (1)
70. if (E is PBe) and (dE is PMdE) then (KK is NL) (1)
71. if (E is PBe) and (dE is PBdE) then (KK is NL) (1)
72. if (E is PBe) and (dE is PLE) then (KK is NL) (1)
73. if (E is PLe) and (dE is NLdE) then (KK is PM) (1)
74. if (E is PLe) and (dE is NBdE) then (KK is PM) (1)
75. if (E is PLe) and (dE is NMdE) then (KK is PB) (1)
76. if (E is PLe) and (dE is NSdE) then (KK is PB) (1)
77. if (E is PLe) and (dE is ZdE) then (KK is NB) (1)
78. if (E is PLe) and (dE is PSdE) then (KK is NL) (1)
79. if (E is PLe) and (dE is PMdE) then (KK is NL) (1)
80. if (E is PLe) and (dE is PBdE) then (KK is NL) (1)
81. if (E is PLe) and (dE is PLE) then (KK is NL) (1)

Налаштовування управляючих правил для даного випадку наведено на рисунку (див. рис. 9.40).

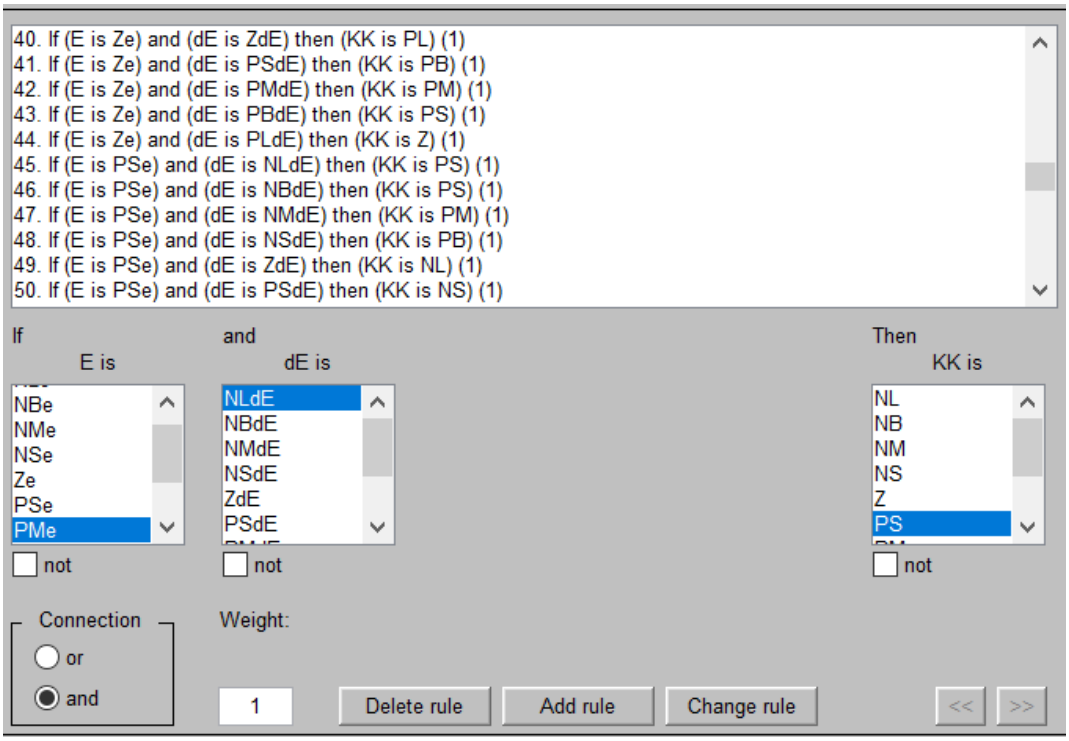


Рисунок 9.40 - Налаштовування опису правил функціонування нечіткого інкрементального супервізору НЛР_ПД

Посилки у правилах пов'язані (connection) за допомогою операції *and*. Введене правило забезпечене ваговим коефіцієнтом (*Weight*=1).

Після опису правил можливо за допомогою

- інтерфейсу *Rule Viewer* можливо переглянути роботу системи нечіткого висновку за різних вхідних даних (див. рис. 9.41).

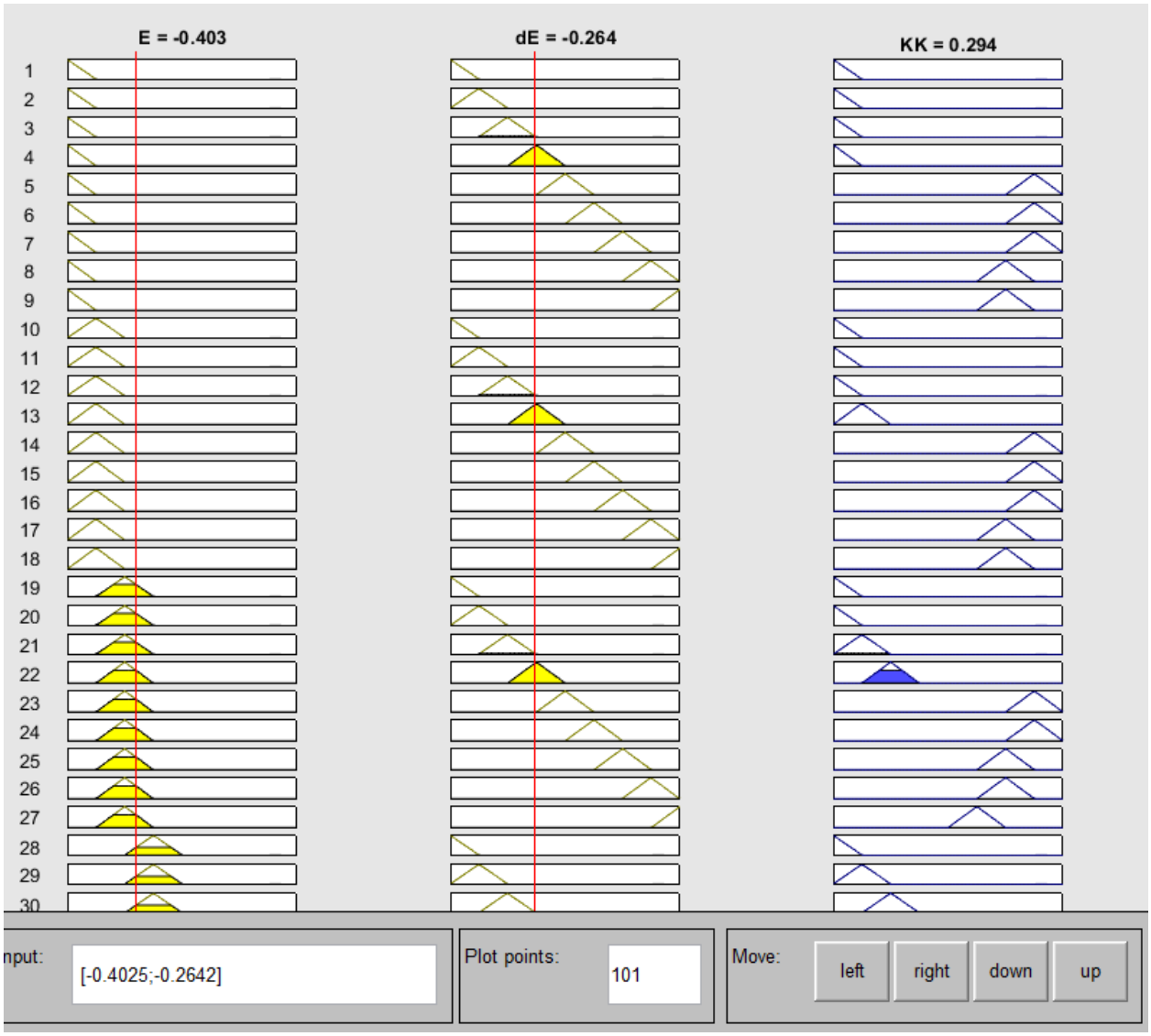


Рисунок 9.41 - Робота системи нечіткого інкрементального супервізору НЛР_ПД

− інтерфейсу Surface Viewer можливо переглянути графічне подання закону управління (рис. 9.42).

Після процедур налаштування нечіткого інкрементального супервізору НЛР_ПД у редакторі системи нечіткого висновку здійснює запис результатів налаштування у Fuzzy Logic Controller. Для цього у вкладці File здійснюємо Export в математичну модуль нечіткого контролера From Workspase вказавши ім’я Fuzzy Logic Controller.

На рис. 9.43 представлені графіки перехідного процесу у системі управління з нечітким інкрементальним супервізором НЛР_ПД. На математичної моделі (див. рис. 9.35) у момент часу 15 с додано збурення. Як видно з графіку перехідного процесу, що спроектована система управління повністю компенсує збурюючи впливи.

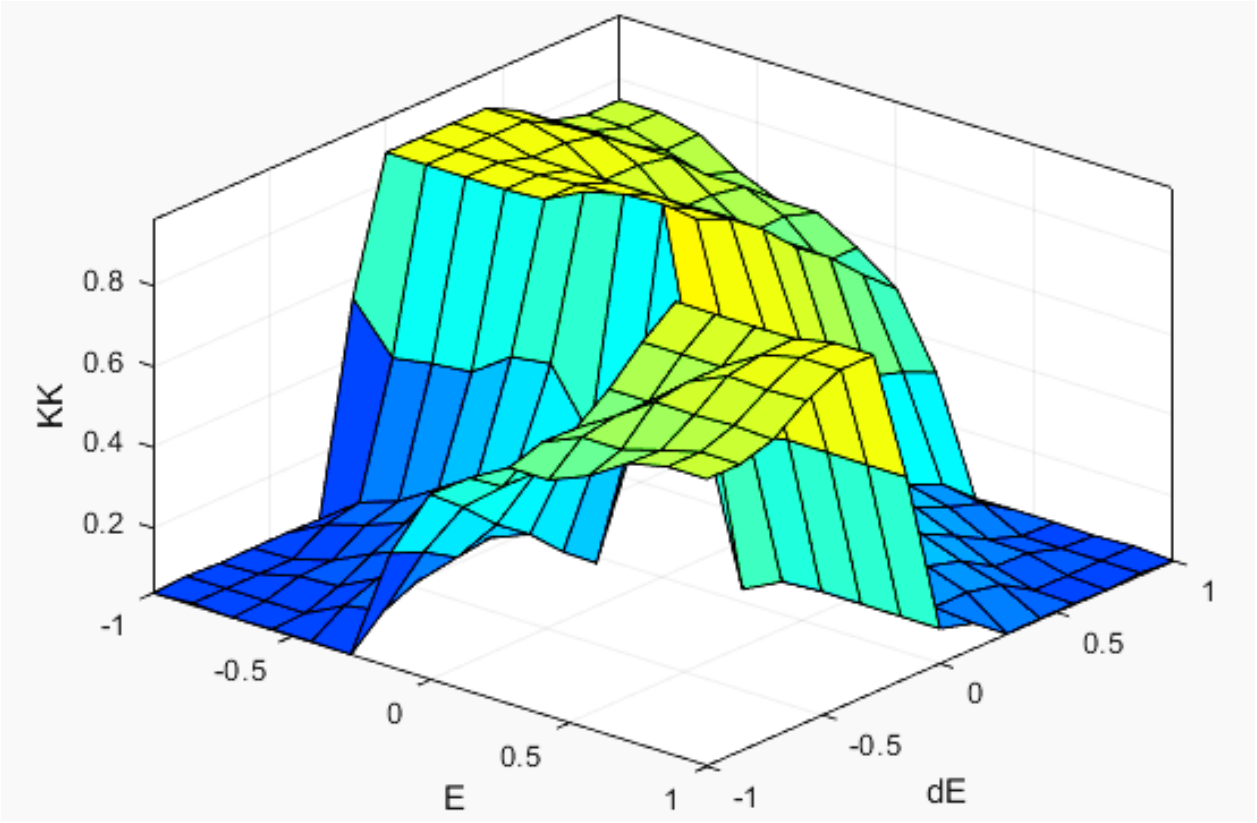


Рисунок 9.42 - Графічне подання закону управління

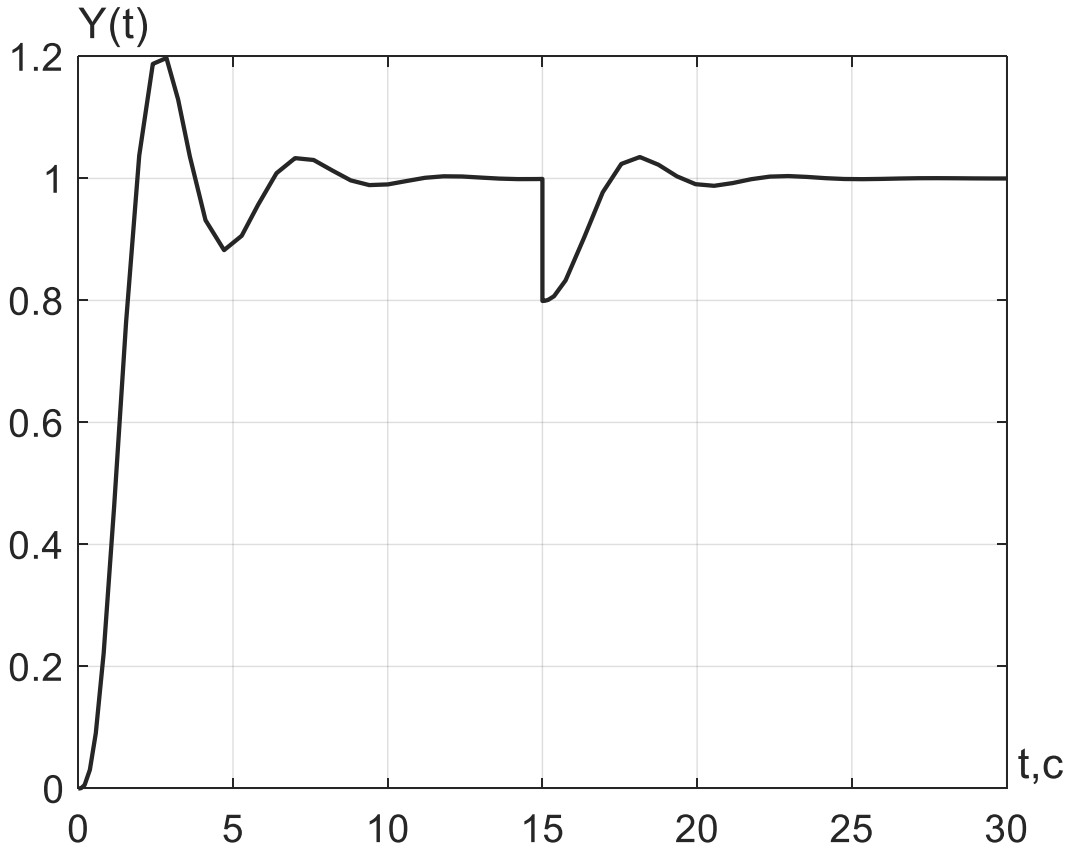


Рисунок 9.43 – Графік перехідного процесу у системі управління з нечітким інкрементальним супервізором ПД-типу

Розглянемо ще один варіант організації нечіткого супервізора, в якому корекція коефіцієнтів ПІД-регулятора відбувається на підставі інформації про помилку та її похідну.

Припустимо, що відомі діапазони, в яких можуть змінюватися коефіцієнти K_p та K_d:

$$K_p \in [K_{p.min}, K_{p.max}],$$

$$K_d \in [K_{d.min}, K_{d.max}]$$

При виборі діапазону зміни коефіцієнтів можна керуватися такими формулами

$$K_{p.min} = 0.32K_u;\ \ K_{p.max} = 0.6K_u$$
$$K_{d.min} = 0.08K_uT_u;\ \ K_{d.max} = 0.15K_uT_u$$

де K_u – мінімальне значення коефіцієнта посилення П–регулятора, у якому виникають автоколивання з періодом T_u.

Вважатимемо, що значення коефіцієнтів нормалізовані:

$$K_p^{/} = \frac{K_p - K_{p.min}}{K_{p.max} - K_{p.min}};\ \ K_d^{/} = \frac{K_d - K_{d.min}}{K_{d.max} - K_{d.min}}$$

відкіля

$$K_p = \left(K_{p.max} - K_{p.min}\right)K_p^{/} + K_{p.min};$$

$$K_d = (K_{d.max} - K_{d.min})K_d^{/} + K_{d.min}.$$

Згідно з методикою Зіглера - Ніколса, постійна часу інтегрування залежить від постійного часу диференціювання:

$$T_i = \alpha T_d,$$

таким чином

$$K_i = \frac{K_p}{\alpha T_d} = \frac{\left(K_p\right)^2}{\alpha K_d}$$

Супервізор містить безліч правил виду:

Якщо ($e(t) = A_i$) та ($de(t)/dt = B_i$), то ($K'_p = C_i$) і ($K'_d = D_i$) та $\alpha = \alpha_i$.;
$i = 1, 2, \ldots m$,

де A_i, B_i, C_i і D_i – нечіткі множини, α_i – константа.
Лінгвістичне опис $e(t)$ та $de(t)/dt$ показано на рис. 9.44.

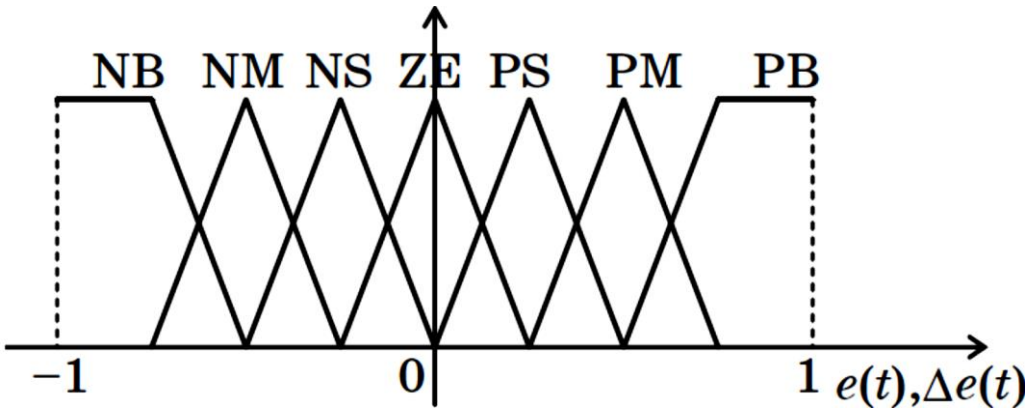


Рисунок 9.44 – Лінгвістичний опис помилки та її прирощення

Лінгвістичний опис K'_p та K'_d показано на рис. 9.45, де використано лише два терми: *Big* та *Small* («велике» і «мале»), для їх опису можуть бути використані дзвонові функції приналежності.

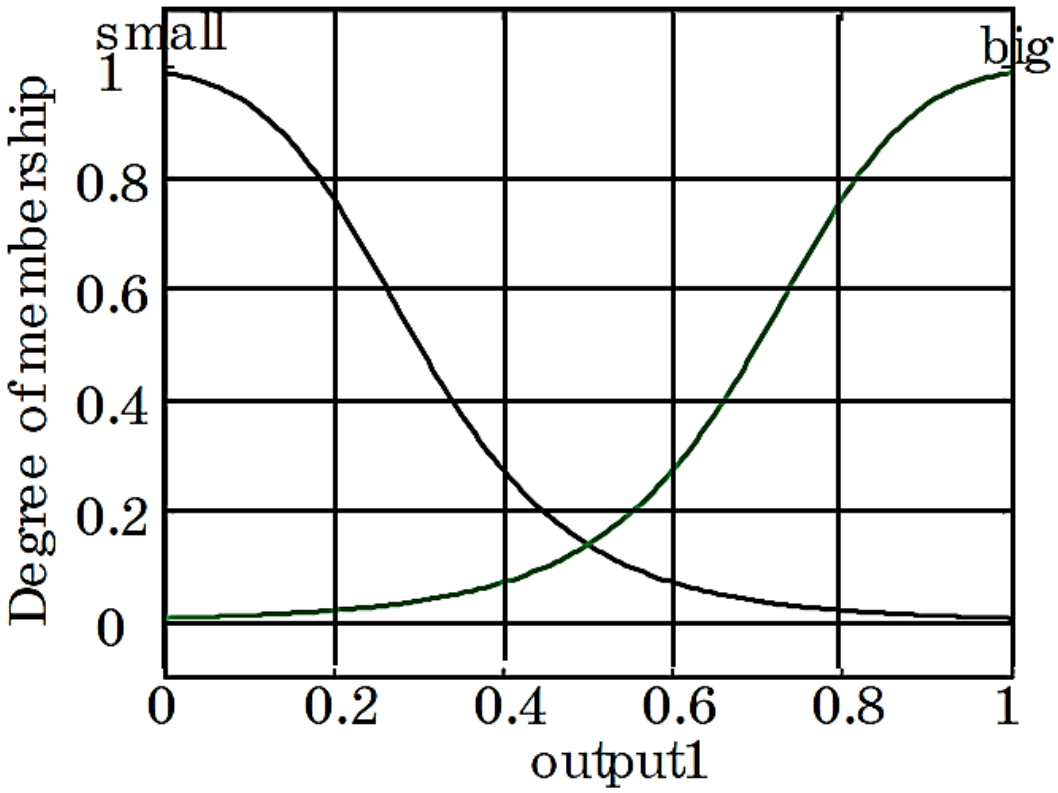


Рисунок 9.45 - Лінгвістичний опис K'p та K'd

Для опису правил зміни коефіцієнтів розглянемо характерні точки типового перехідного процесу (рис. 9.46).

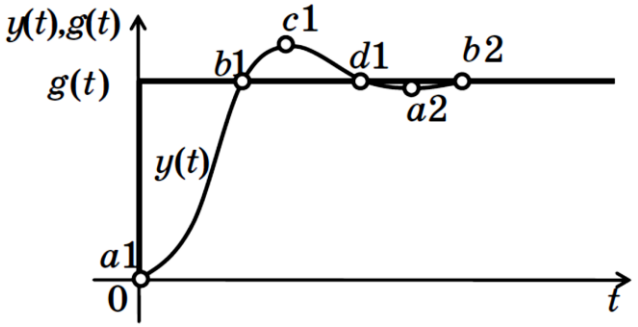


Рисунок 9.46 – Бажаний перехідний процес

На початку перехідного процесу (точка a_1) потрібно велике значення сигналу керування для отримання малого часу наростання. Тому ПІД-регулятор повинен мати велике значення K_p та K_i, а значення K_d має бути малим, тобто K'_p повинен мати значення *Big*, а K'_d – *Small*.

Для забезпечення великого значення K_i (порівняно з вихідним значенням, отриманим за методом Зіглера – Ніколса) треба, щоб значення α було мало.

У методі Зіглера – Ніколса T_i задається вчетверо більше T_d ($\alpha = 4$).

Тому якщо α набуває значення менше 4, то вплив інтеграла посилюється, таким чином, у точці a_1 справедливо правило виду:

Якщо ($e(k)$ = PB) та ($\Delta e(k)$ = ZE), то (K'_p = *Big*) та (K'_d = *Small*) та $\alpha = 2$.

Величина a може розглядатися як синглетон (рис. 9.47), де використані позначення *S-small*, *MS-medium small, M-medium, B-big*).

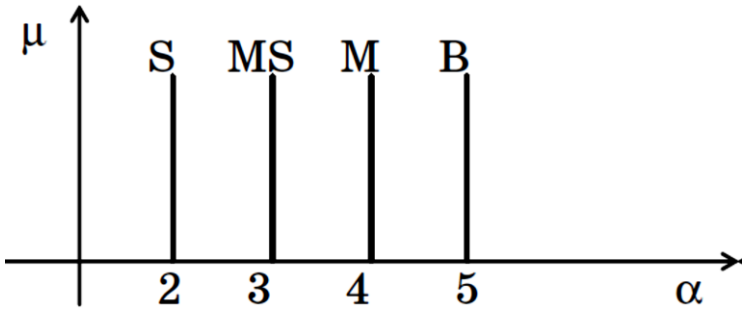


Рисунок 9.47 – Лінгвістичний опис варіантів значень α

В околиці точки b_1 (див. рис. 9.46) сигнал керування повинен бути малим для уникнення великого перерегулювання. Тому ПІД-регулятор повинен мати малі K_p та K_i і великий K_d. Ці умови описує правило:

Якщо ($e(k)$ = ZE) та ($\Delta e(k)$ = NB), то (K'_p = Small) та (K'_d = Big) та $\alpha = 5$.

Повторюючи наведені міркування, можна описати правила зміни K_p та K_d у вигляді табл. 9.6 та 9.7.

Таблиця 9.6 – Правила визначення K'_p

Таблиця правил		*de*(*t*)/*dt*						
		NB	*NM*	*NS*	*ZE*	*PS*	*PM*	*PB*
e(*t*)	*NB*	**B**	**B**	**B**	**B**	**B**	**B**	**B**
	NM	**S**	**B**	**B**	**B**	**B**	**B**	**S**
	NS	**S**	**S**	**B**	**B**	**B**	**S**	**S**
	ZE	**S**	**S**	**S**	**S**	**S**	**S**	**S**
	PS	**S**	**S**	**S**	**B**	**B**	**S**	**S**
	PM	**S**	**B**	**B**	**B**	**B**	**B**	**S**
	PB	**B**	**B**	**B**	**B**	**B**	**B**	**B**

K'_P

Таблиця 9.7 – Правила визначення K'_d

Таблиця правил		*de*(*t*)/*dt*						
		NB	*NM*	*NS*	*ZE*	*PS*	*PM*	*PB*
e(*t*)	*NB*	**S**	**S**	**S**	**S**	**S**	**S**	**S**
	NM	**B**	**B**	**S**	**S**	**S**	**B**	**B**
	NS	**B**	**B**	**B**	**S**	**B**	**B**	**B**
	ZE	**B**	**B**	**B**	**B**	**B**	**B**	**B**
	PS	**B**	**B**	**B**	**S**	**B**	**B**	**B**
	PM	**B**	**B**	**S**	**S**	**S**	**B**	**B**
	PB	**S**	**S**	**S**	**S**	**S**	**S**	**S**

K'_D

Закон зміни коефіцієнта a показаний у табл. 9.8.

Таблиця 9.8 – Правила визначення α

Таблиця правил		*de*(*t*)/*dt*						
		NB	*NM*	*NS*	*ZE*	*PS*	*PM*	*PB*
e(*t*)	*NB*	**2**	**2**	**2**	**2**	**2**	**2**	**2**
	NM	**3**	**3**	**2**	**2**	**2**	**3**	**3**
	NS	**4**	**3**	**3**	**2**	**3**	**3**	**4**
	ZE	**5**	**4**	**3**	**3**	**3**	**4**	**5**
	PS	**4**	**3**	**3**	**2**	**3**	**3**	**4**
	PM	**3**	**3**	**2**	**2**	**2**	**3**	**3**
	PB	**2**	**2**	**2**	**2**	**2**	**2**	**2**

a

Таблиці 9.6–9.8 можуть бути зведені в одну таблицю з 49 правил, що мають дві посилки та три висновки.

Приклад. На рис. 9.45 показана схема моделювання супервізора Simulink MatLab для об'єкта, заданого передатною функцією:

$$W(p) = \frac{27}{(p+1)(p+3)^3}.$$

Fuzzy Logic Controller надаємо Fis name: PIDsupv3 (ім’я може буди будь-яке).

Викликається редактор системи нечіткого висновку (Fuzzy Inference System – FIS) у MatLab командою

>> fuzzy

Дале проводиться налаштовування головного вікна редактора нечіткої логічної системи (див. рис. 9.46).

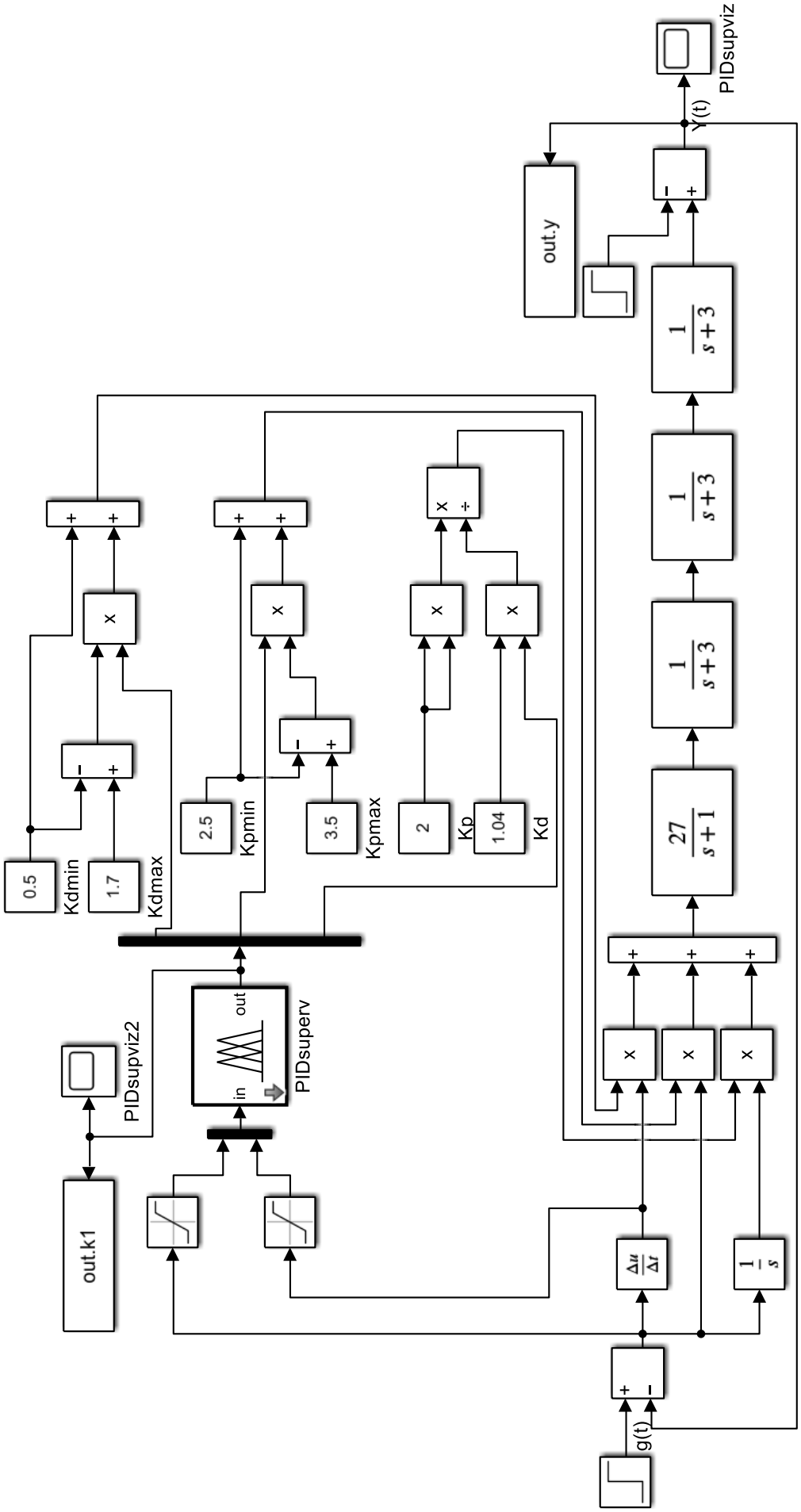


Рисунок 9.45 – Структурна схема математичної моделі системи управління

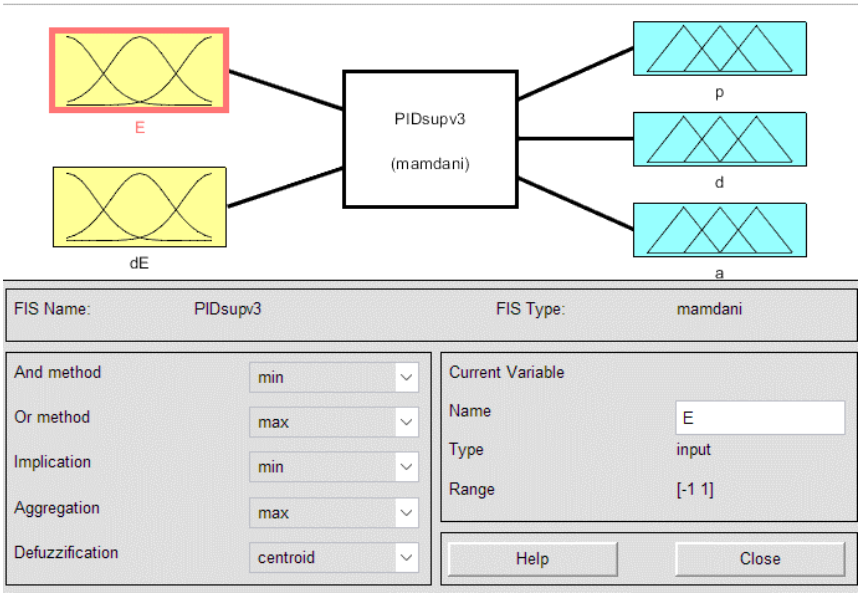


Рисунок 9.46 – Налаштування інтерфейсу FIS editor

У *FIS editor* задається опис систему нечіткого логічного висновку *Mamdani*. Для створюваної системи обирається вид логічного зв'язку (*And method – min*) та (*Or method – max*), вид імплікації (*Implication – min*), спосіб агрегування висновків правил (*Aggregation – max*) та метод дефазифікації (*Defuzzification – centroid*).

У меню *Edit* послідовно додаємо 2 вхідні змінні з розміром базової шкали (Range= [-1 1]) та 3 вихідні змінні з розміром базової шкали (*Range*= [0 1]).

Для опису вхідних логічних змінних у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної змінної трикутну функцію приналежності.

Нехай терми лінгвістичної змінної «Помилка управління» розміщуються відповідно до рис. 9.47 з 7 параметрами функцій приналежності:

Name = "NBe"; Type = "trimf"; Param = [-1.33 -1 -0.66];
Name = "NMe"; Type = "trimf"; Param = [-1 -0.66 -0.33];
Name = "NSe"; Type = "trimf"; Param = [-0.66 -0.33 0];
Name = "Ze"; Type = " trimf"; Param = [-0.33 0 0.33];
Name = "PSe"; Type = "trimf"; Param = [0 0.33 0.66].
Name = "PMe"; Type = "trimf"; Param = [0.33 0.66 1].
Name = "PBe"; Type = "trimf"; Param = [0.66 1 1.33].

Терми вхідної лінгвістичної змінної «Похідна помилки управління» формуємо рівномірно. Для опису вхідних логічних змінних у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної вихідної змінної трикутну функцію приналежності. З параметрами:

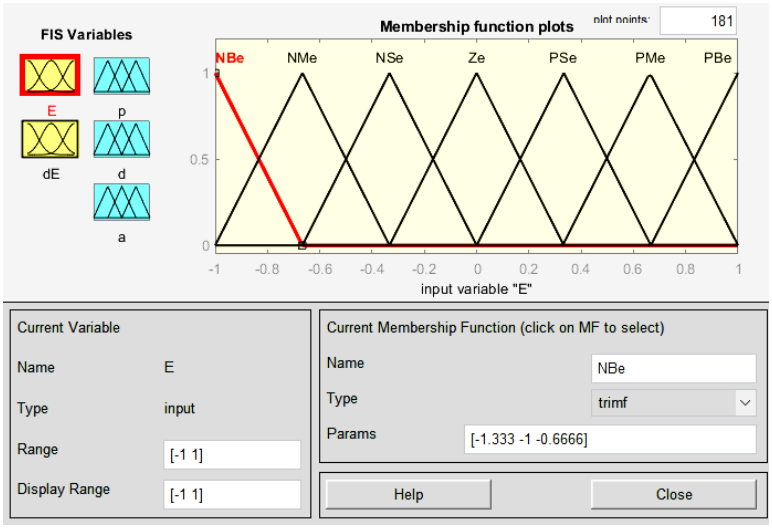


Рисунок 9.47 – Результат налаштування функцій приналежності вхідних змінних «Помилка управління»

Name = "NBde"; Type = "trimf"; Param = [-1.33 -1 -0.66];
Name = "NMde"; Type = "trimf"; Param = [-1 -0.66 -0.33];
Name = "NSde"; Type = "trimf"; Param = [-0.66 -0.33 0];
Name = "Zde"; Type = " trimf"; Param = [-0.33 0 0.33];
Name = "PSde"; Type = "trimf"; Param = [0 0.33 0.66].
Name = "PMde"; Type = "trimf"; Param = [0.33 0.66 1].
Name = "PBde"; Type = "trimf"; Param = [0.66 1 1.33].

Результат налаштування функцій приналежності вхідних змінних «Похідна помилки управління» на рис. 9.48.

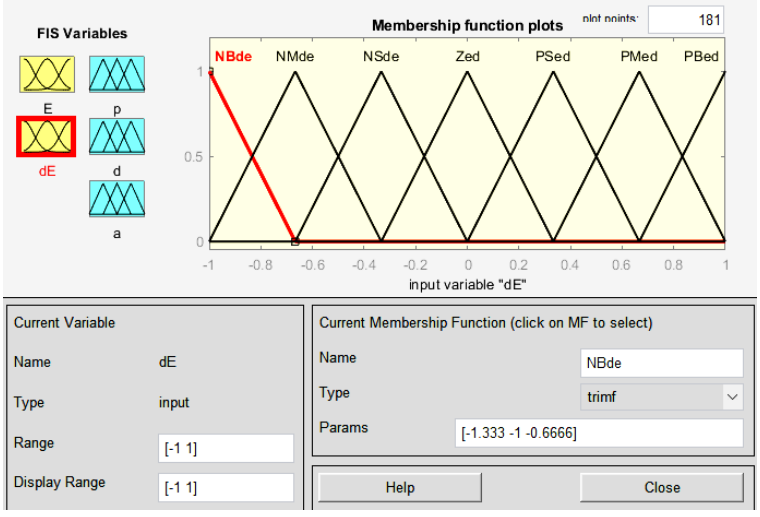


Рисунок 9.48 – Результат налаштування функцій приналежності вхідних змінних «Похідна помилки управління»

Для опису вихідної логічної змінної K'_p у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної вихідної змінної дзвоноподібну функцію приналежності.

Name = "Sp"; Type = "gbellmf"; Param = [0.5 2.5 0];
Name = "Bp"; Type = "gbellmf"; Param = [0.5 2.5 1].

Результат налаштування функцій приналежності вхідних змінних наведено на рис. 9.49.

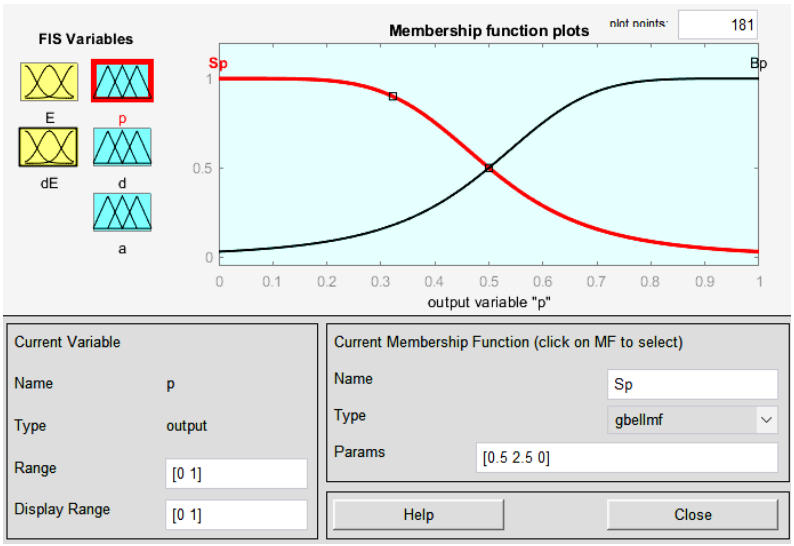


Рисунок 9.49 – Результат налаштування функцій приналежності вихідної змінної K'_p

Для опису вихідної логічної змінної K'_d у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної вихідної змінної дзвоноподібну функцію приналежності.

Name = "Sd"; Type = " trimf "; Param = [0.5 2.5 0];
Name = "Bd"; Type = " trimf "; Param = [0.5 2.5 1].

Результат налаштування функцій приналежності вхідних змінних наведено на рис. 9.50.

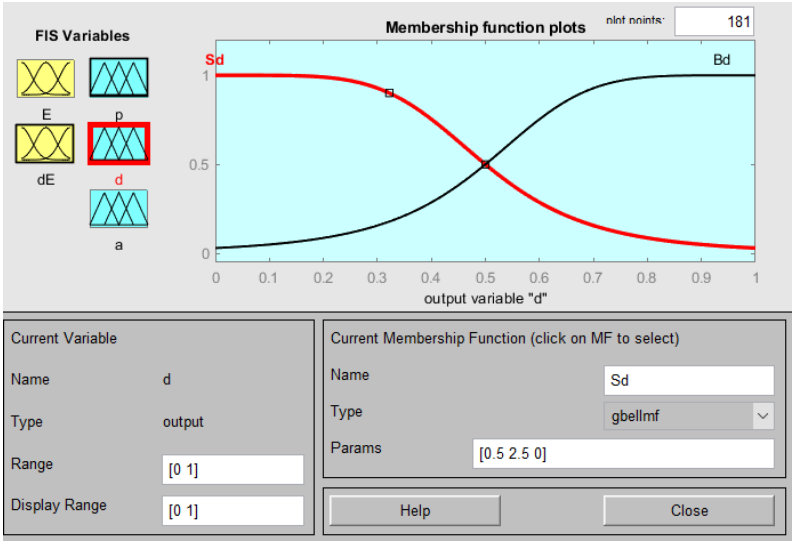


Рисунок 9.50 – Результат налаштування функцій приналежності вихідної змінної K'_d

Для опису вихідної логічної змінної α у редакторі функцій приналежності (*Membership Function Editor*), задаємо для кожної вихідної змінної трикутну функцію приналежності.

Name = "Sa"; Type = "gbellmf"; Param = [0.5 2.5 0];

Name = "Ba"; Type = "gbellmf"; Param = [0.5 2.5 1].

Результат налаштування функцій приналежності вхідних змінних наведено на рис. 9.51.

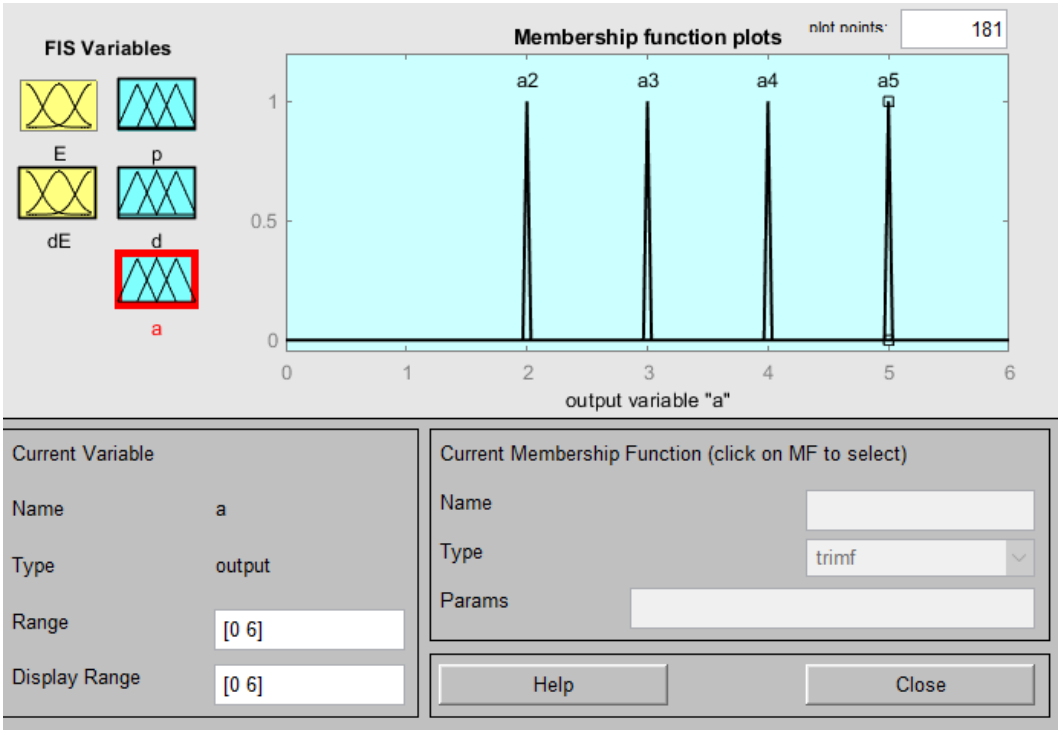


Рисунок 9.51 – Результат налаштування функцій приналежності вихідної змінної α

Після опису вхідні та вихідні лінгвістичних змінних, здійснюється опис правил у вікні редактора Rule Editor, кількість правил для даного випадку 25. Управляючі правила сформульовано згідно з табл. 9.6-9.8:

1. if (E is NBe) and (dE is NBdE) then (p is Bp) (d is Sd) (a is a2) (1)
2. if (E is NBe) and (dE is NMdE) then (p is Bp) (d is Sd) (a is a2) (1)
3. if (E is NBe) and (dE is NSdE) then (p is Bp) (d is Sd) (a is a2) (1)
4. if (E is NBe) and (dE is ZdE) then (p is Bp) (d is Sd) (a is a2) (1)
5. if (E is NBe) and (dE is PSdE) then (p is Bp) (d is Sd) (a is a2) (1)
6. if (E is NBe) and (dE is PMdE) then (p is Bp) (d is Sd) (a is a2) (1)
7. if (E is NBe) and (dE is PBdE) then (p is Bp) (d is Sd) (a is a2) (1)
8. if (E is NMe) and (dE is NBdE) then (p is Sp) (d is Bd) (a is a3) (1)
9. if (E is NMe) and (dE is NMdE) then (p is Bp) (d is Bd) (a is a3) (1)
10. if (E is NMe) and (dE is NSdE) then (p is Bp) (d is Sd) (a is a2) (1)
11. if (E is NMe) and (dE is ZdE) then (p is Bp) (d is Sd) (a is a2) (1)
12. if (E is NMe) and (dE is PSdE) then (p is Bp) (d is Sd) (a is a2) (1)
13. if (E is NMe) and (dE is PMdE) then (p is Bp) (d is Bd) (a is a3) (1)

14. if (E is NMe) and (dE is PBdE) then (p is Sp) (d is Bd) (a is a3) (1)
15. if (E is NSe) and (dE is NBdE) then (p is Sp) (d is Bd) (a is a4) (1)
16. if (E is NSe) and (dE is NMdE) then (p is Sp) (d is Bd) (a is a3) (1)
17. if (E is NSe) and (dE is NSdE) then (p is Bp) (d is Bd) (a is a3) (1)
18. if (E is NSe) and (dE is ZdE) then (p is Bp) (d is Sd) (a is a2) (1)
19. if (E is NSe) and (dE is PSdE) then (p is Bp) (d is Bd) (a is a3) (1)
20. if (E is NSe) and (dE is PMdE) then (p is Sp) (d is Bd) (a is a3) (1)
21. if (E is NSe) and (dE is PBdE) then (p is Sp) (d is Bd) (a is a4) (1)
22. if (E is Ze) and (dE is NBdE) then (p is Sp) (d is Bd) (a is a5) (1)
23. if (E is Ze) and (dE is NMdE) then (p is Sp) (d is Bd) (a is a4) (1)
24. if (E is Ze) and (dE is NSdE) then (p is Sp) (d is Bd) (a is a3) (1)
25. if (E is Ze) and (dE is ZdE) then (p is Bp) (d is Bd) (a is a3) (1)
26. if (E is Ze) and (dE is PSdE) then (p is Sp) (d is Bd) (a is a3) (1)
27. if (E is Ze) and (dE is PMdE) then (p is Sp) (d is Bd) (a is a4) (1)
28. if (E is Ze) and (dE is PBdE) then (p is Sp) (d is Bd) (a is a5) (1)
29. if (E is PSe) and (dE is NBdE) then (p is Sp) (d is Bd) (a is a4) (1)
30. if (E is PSe) and (dE is NMdE) then (p is Sp) (d is Bd) (a is a3) (1)
31. if (E is PSe) and (dE is NSdE) then (p is Bp) (d is Bd) (a is a3) (1)
32. if (E is PSe) and (dE is ZdE) then (p is Bp) (d is Sd) (a is a2) (1)
33. if (E is PSe) and (dE is PSdE) then (p is Bp) (d is Bd) (a is a3) (1)
34. if (E is PSe) and (dE is PMdE) then (p is Sp) (d is Bd) (a is a3) (1)
35. if (E is PSe) and (dE is PBdE) then (p is Sp) (d is Bd) (a is a4) (1)
36. if (E is PMe) and (dE is NBdE) then (p is Sp) (d is Bd) (a is a3) (1)
37. if (E is PMe) and (dE is NMdE) then (p is Bp) (d is Bd) (a is a3) (1)
38. if (E is PMe) and (dE is NSdE) then (p is Bp) (d is Sd) (a is a2) (1)
39. if (E is PMe) and (dE is ZdE) then (p is Bp) (d is Sd) (a is a2) (1)
40. if (E is PMe) and (dE is PSdE) then (p is Bp) (d is Sd) (a is a2) (1)
41. if (E is PMe) and (dE is PMdE) then (p is Bp) (d is Bd) (a is a3) (1)
42. if (E is PMe) and (dE is PBdE) then (p is Sp) (d is Bd) (a is a3) (1)
43. if (E is PBe) and (dE is NBdE) then (p is Bp) (d is Sd) (a is a2) (1)
44. if (E is PBe) and (dE is NMdE) then (p is Bp) (d is Sd) (a is a2) (1)
45. if (E is PBe) and (dE is NSdE) then (p is Bp) (d is Sd) (a is a2) (1)
46. if (E is PBe) and (dE is ZdE) then (p is Bp) (d is Sd) (a is a2) (1)
47. if (E is PBe) and (dE is PSdE) then (p is Bp) (d is Sd) (a is a2) (1)
48. if (E is PBe) and (dE is PMdE) then (p is Bp) (d is Sd) (a is a2) (1)
49. if (E is NBe) and (dE is PBdE) then (p is Bp) (d is Sd) (a is a2) (1)

Налаштовування управляючих правил для даного випадку наведено на рисунку (див. рис. 9.52).

Посилки у правилах пов'язані (connection) за допомогою операції *and*. Введене правило забезпечене ваговим коефіцієнтом (*Weight*=1).

Після опису правил можливо за допомогою

- інтерфейсу *Rule Viewer* можливо переглянути роботу системи нечіткого висновку за різних вхідних даних (див. рис. 9.53).

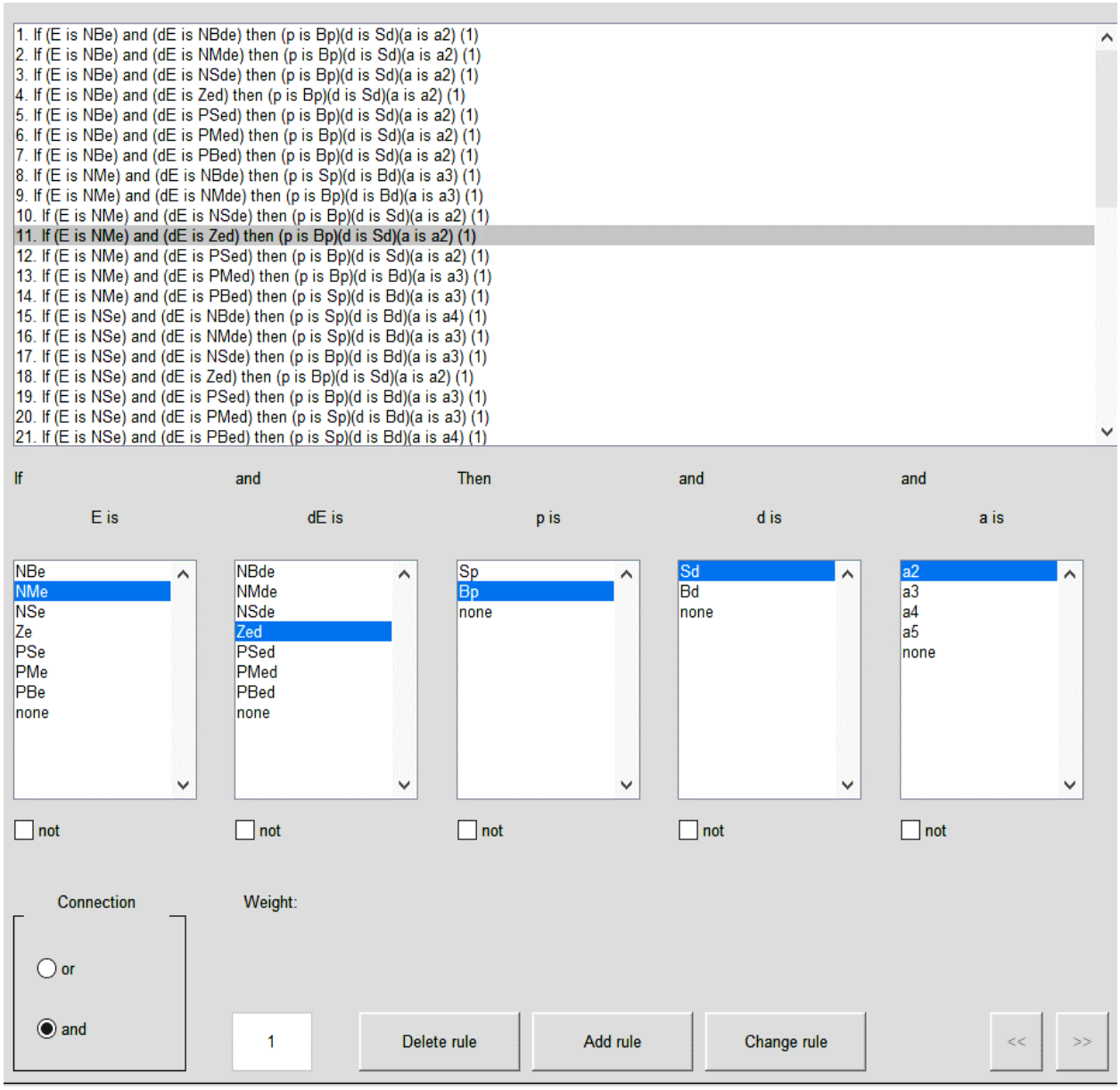


Рисунок 9.52 - Налаштовування опису правил функціонування нечіткого супервізору НЛР_ПІД

− інтерфейсу Surface Viewer можливо переглянути графічне подання закону управління (рис. 9.54).

Після процедур налаштування нечіткого супервізору НЛР_ПІД у редакторі системи нечіткого висновку здійснює запис результатів налаштування у Fuzzy Logic Controller. Для цього у вкладці File здійснюємо Export в математичну модуль нечіткого контролера From Workspase вказавши ім’я Fuzzy Logic Controller.

На рис. 9.55 представлені графіки перехідного процесу у системі управління з нечітким супервізором НЛР_ПІД. На математичної моделі (див. рис. 9.45) у момент часу 12 с додано збурення. Як видно з графіку перехідного процесу, що спроектована система управління повністю компенсує збурюючи впливи .

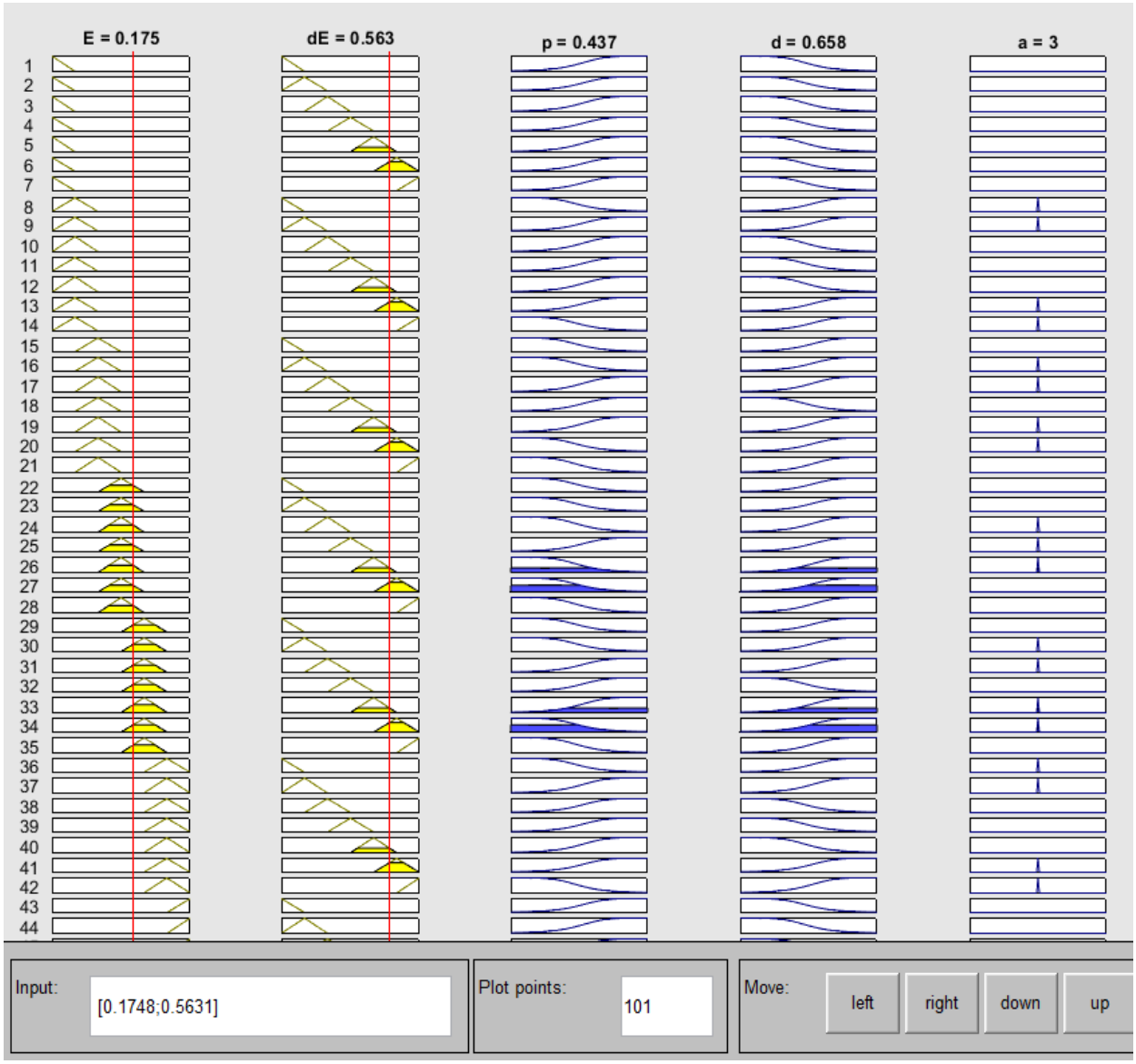


Рисунок 9.53 - Робота системи нечіткого супервізору НЛР_ПІД

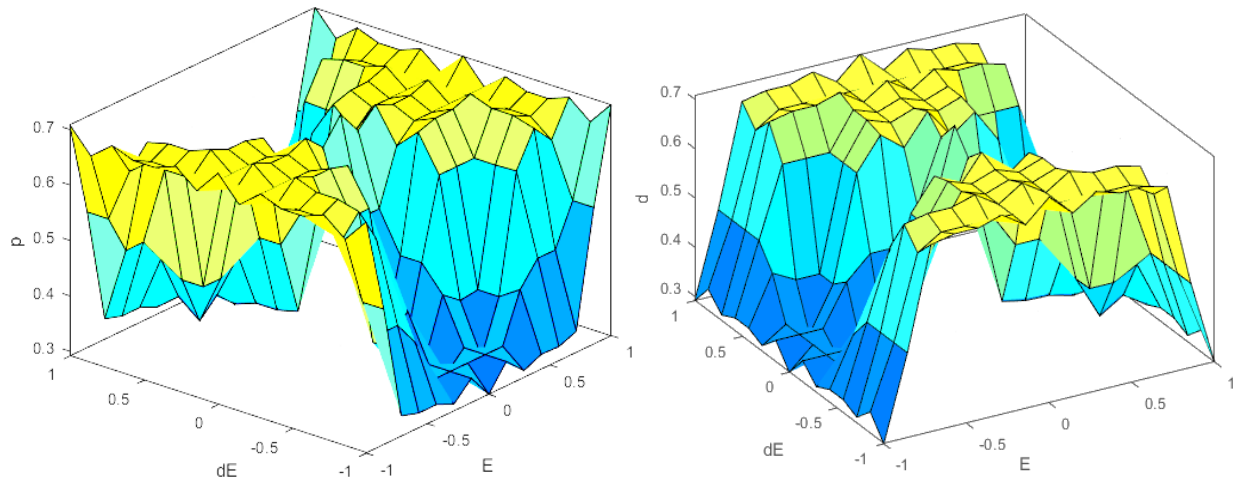


Рисунок 9.54 - Графічне подання закону управління K'_p, K'_d

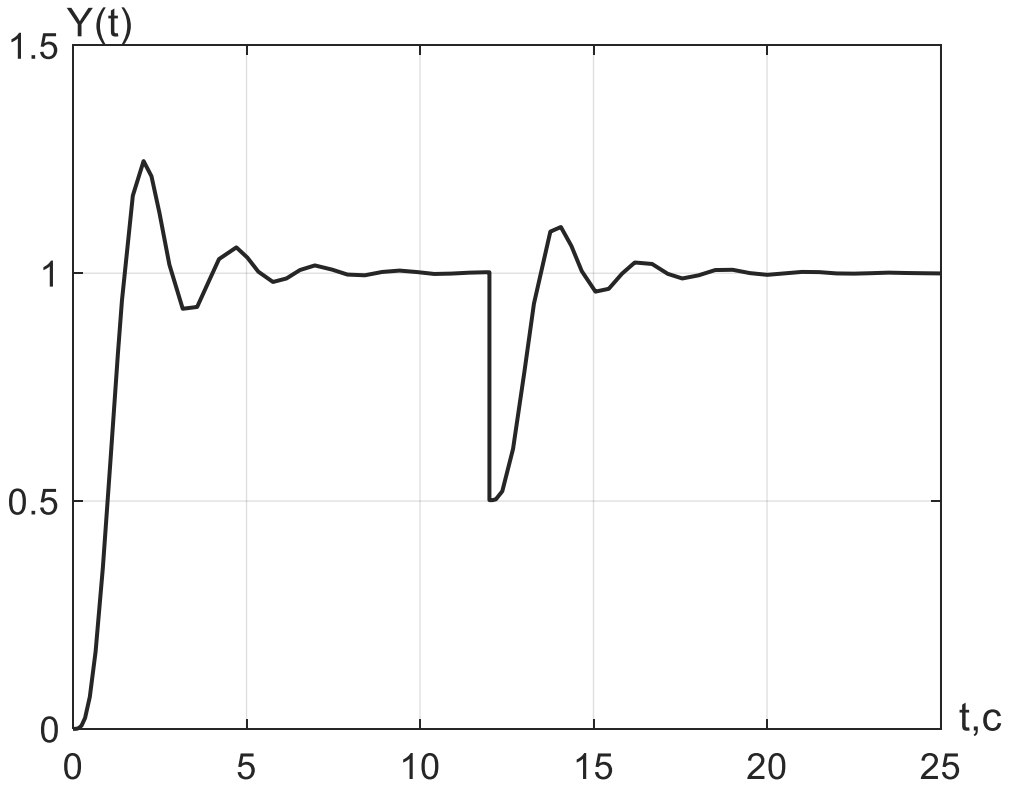


Рисунок 9.55 – Графік перехідного процесу у системі управління з нечітким супервізором ПІД-типу

ПІД регулятори набули широкого поширення в промисловості завдяки простоті своєї структури та надійності в широкому діапазоні робочих умов. Розробка цих регуляторів пов'язана з вибором трьох параметрів: пропорційного коефіцієнта, коефіцієнтів при інтегралі та похідній.

Існуючі ПІД-регулятори можна розбити на два класи: регулятори з фіксованим налаштуванням та адаптивні ПІД-регулятори.

Регулятори з фіксованим налаштуванням налаштовуються одноразово, наприклад, за алгоритмом Зіглера – Ніколса.

В адаптивних ПІД-регуляторах коефіцієнти можуть змінюватися в реальному часі, що вимагає використання додаткових знань про процес управління.

Розглянуті варіанти нечітких супервізорів реалізують прості алгоритми адаптації на підставі оцінки помилки керування та її похідної.

Використана література

1 . Albus J. S., Meystel A. M. Intelligent Systems: Architecture, Design, and Control / Wiley, New York, 2002.

2 A. P. Engelbrecht. Computational Intelligence: An Introduction / Wiley, Chichester, U.K., 2002.

3 Badiru A. B., Cheung J. Y. Fuzzy Engineering Expert Systems with Neural Network Applications / John Wiley, New York, NY, 2002.

4 Апостолюк В. О. Інтелектуальні системи керування: конспект лекцій [Текст] / В. О. Апостолюк, О. С. Апостолюк. – К.: НТУУ «КПІ», 2008. – 88 с.

5 Антоненко В. М. Сучасні інформаційні системи і технології: управління знаннями : навчальний посібник / В. М. Антоненко, С. Д. Мамченко, Ю. В. Рогушина. – Ірпінь : Національний університет ДПС України, 2016. – 212 с. ISBN 978-966-337-418-5

6 Інтелектуальні системи управління: Експертні системи - основи проектування та застосування в системах автоматизації [Електронний ресурс] : навч. посіб. для студ. спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» / КПІ ім. Ігоря Сікорського; уклад.: Л. Д. Ярощук. – Електронні текстові дані (1 файл: 2,56 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2019. – 136с.

7 Giese H., Rumpe B. Science and Engineering of Cyber-Physical Systems (Dagstuhl Seminar 11441), Dagstuhl Reports, vol. 1, no. 11, pp. 1–22, 2012.

8 Conti M. Looking ahead in pervasive computing: challenges and opportunities in the era of cyber-physical convergence," Pervasive and Mobile Computing, 2011.

9 Sha L., Gopalakrishnan S. Cyber-physical systems: A new frontier, Machine Learning in Cyber Trust, pp. 3–13, 2009.

10 Horv´ath I., Gerritsen B. Cyber-physical systems: Concepts, technologies and implementation principles, in Tools and Methods of Competitive Engineering Symposium (TMCE), 2012, pp. 19–36.

11 Lee E., "Computing needs time," Communications of the ACM, vol. 52, no. 5, pp. 70–79, 2009.