МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО
ІНСТИТУТ ЕЛЕКТРОМЕХАНІКИ, ЕНЕРГОЗБЕРЕЖЕННЯ
І СИСТЕМ УПРАВЛІННЯ

МЕТОДИЧНІ ВКАЗІВКИ
ЩОДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
«ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ КЕРУВАННЯ ЕЛЕКТРОМЕХАНІЧНИМИ СИСТЕМАМИ»

ДЛЯ СТУДЕНТІВ ДЕННОЇ ТА ЗАОЧНОЇ ФОРМ НАВЧАННЯ
ЗІ СПЕЦІАЛЬНОСТІ
141 «ЕЛЕКТРОЕНЕРГЕТИКА, ЕЛЕКТРОТЕХНІКА
ТА ЕЛЕКТРОМЕХАНІКА»
ОСВІТНЬО-НАУКОВОЇ ПРОГРАМ
«ЕЛЕКТРОМЕХАНІЧНІ СИСТЕМИ АВТОМАТИЗАЦІЇ
ТА ЕЛЕКТРОПРИВОД»

КРЕМЕНЧУК 2018

ЗМІСТ

ВСТУП

Необхідністю вивчення дисципліни «Інтелектуальні системи керування електромеханічними системами» є підготовка фахівців, здатних самостійно і творчо розв’язувати задачі проектування, дослідження, експлуатації систем керування й автоматики та їх програмного забезпечення.

Метою вивчення дисципліни «Інтелектуальні системи керування електромеханічними системами» є підготовка фахівців у галузі електроенергетики, електротехніки та електромеханіки щодо автоматизації завдань, що складно формалізуються, які до цих пір уважаються прерогативою людини. Завданням вивчення дисципліни є отримання знань про способи мислення людини, а також про методи їх реалізації на комп'ютері.

Мета проведення лабораторних робіт – закріпити знання, отримані на лекціях, шляхом створення прикладних комп'ютерних програм.

Задачі проведення лабораторних робіт, унаслідок яких студенти повинні вміти:

– використовувати знання мов програмування для конкретних технічних задач, набути навичок розробки програмного середовища;

– освоїти роботу з математичним пакетом MatLab, а саме з пакетом розширення з нейронних мереж Neural Networks Toolbox, пакетом нечіткої логіки Fuzzy Logic Toolbox та пакетом генетичних алгоритмів Genetic Algorithm and Direct Search Toolbox.

Методичні вказівки щодо виконання лабораторних робіт з навчальної дисципліни «Інтелектуальні системи керування електромеханічними системами» містять короткі теоре-тичні відомості з кожної теми, порядок виконання роботи, зміст звіту, контрольні питання та вказівки для користування літературними джерелами.

ПЕРЕЛІК ЛАБОРАТОРНИХ РОБІТ

Лабораторна робота № 1

Тема. Створення та настройка експертної системи з використанням Fuzzy Logic Toolbox математичного пакета MatLab

Мета: вивчити основи програмування в математичному пакеті MatLab у додатку Fuzzy Logic Toolbox. Навчитися створювати елементарні експертні системи з набором правил у базі даних системи.

Короткі теоретичні відомості

Fuzzy Logic Toolbox – це пакет прикладних програм, що входять до складу середовища MatLab. Він дозволяє створювати системи нечіткого логічного виведення і нечіткої класифікації в рамках середовища MatLab з можливістю їх інтеграції в Simulink.

Основні властивості:

- визначення змінних, нечітких правил і функцій належності;
- інтерактивний перегляд нечіткого логічного виведення;
- сучасні методи: адаптивне нечітке виведення з використанням нейронних мереж, нечітка кластеризація;
- інтерактивне динамічне моделювання в Simulink;
- генерація переносного *C* коду за допомогою Real-Time Workshop.

Пакет *Fuzzy Logic* містить п’ять графічних редакторів для представлення необхідної інформації в процесі проектування, створення і тестування нечітких моделей.

Пакет *Fuzzy Logic* містить сучасні методи нечіткого моделювання, включаючи:

- адаптивне нечітке виведення з використанням нейронних мереж для автоматичного формування функції належності в процесі навчання їх на вхідних даних;
- нечітку логіку і кластеризацію для задач розпізнавання образів;

– можливість вибору широко відомого метода Мамдані або метода Сугено для створення гібридних нечітких систем.

Пакет дозволяє роботу:

– у режимі графічного інтерфейсу;

– у режимі командного рядка;

– з використанням блоків та прикладів пакета Simulink.

Базовим поняттям *Fuzzy Logic Toolbox* є *FIS-структура* – система нечіткого виведення (*Fuzzy Inference System*). *FIS-структура* містить усі необхідні дані для реалізації функціонального відображення "входи-виходи" на основі нечіткого логічного виведення згідно зі схемою, наведеною на рис. 1.1.

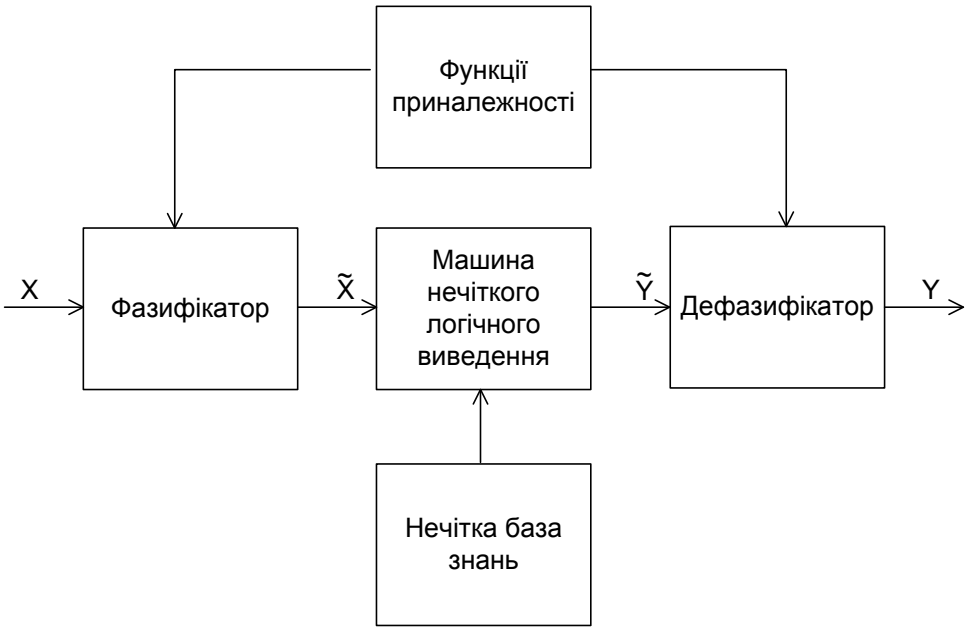


Рисунок 1.1 – Нечітке логічне виведення

Позначення: X – вхідний чіткий вектор; $\tilde{X}$ – вектор нечітких множин, що відповідає вхідному вектору X; $\tilde{Y}$ – результат логічного виведення у вигляді вектора нечітких множин; Y – вихідний чіткий вектор.

Склад графічного інтерфейсу

Fuzzy Logic Toolbox містить наступні редактори:

– редактор нечіткої системи виведення *Fuzzy Inference System Editor* (*FIS Editor* або *FIS-редактор*) разом з додатковими програмами – редактором функцій належності (*Memberhip Function Editor*), редактором правил (*Rule Editor*),

вікно перегляду правил (*Rule Viewer*) і вікном перегляду поверхні відгуку (*SurfaceViewer*);

– редактор гібридних систем (*ANFIS Editor, ANFIS-редактор*);

– програма знаходження кластерів (програма *Clustering* – кластеризація).

Редактор нечіткої системи виведення

Командою (функцією) *fuzzy* з режиму командного рядка запускається основна інтерфейсна програма пакета *Fuzzy Logic* – редактор нечіткої системи виведення. Головне вікно наведено на рисунку 1.2.

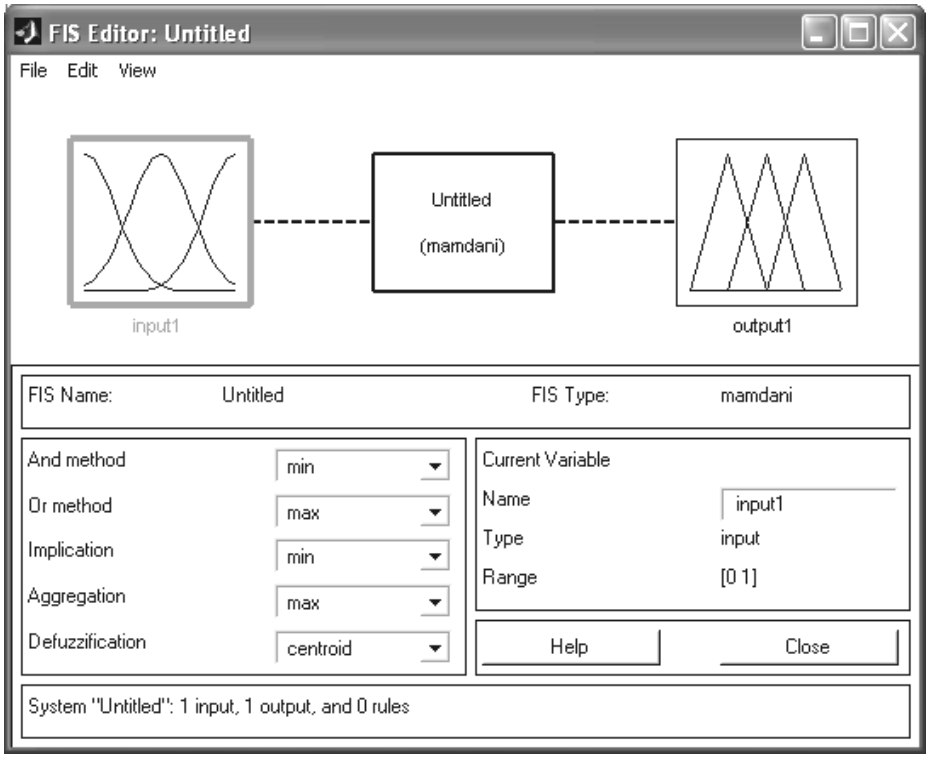


Рисунок 1.2 – Вигляд вікна *FIS Editor*

Графічний інтерфейс гібридних мереж

Головне вікно редактора *ANFIS Editor* викликається командою *anfisedit* з командного рядка, вигляд якого наведено на рисунку 1.3.

За допомогою даного редактора виконується створення або завантаження гібридної системи, перегляд структури, настроювання її параметрів, перевірка якості функціонування такої системи.

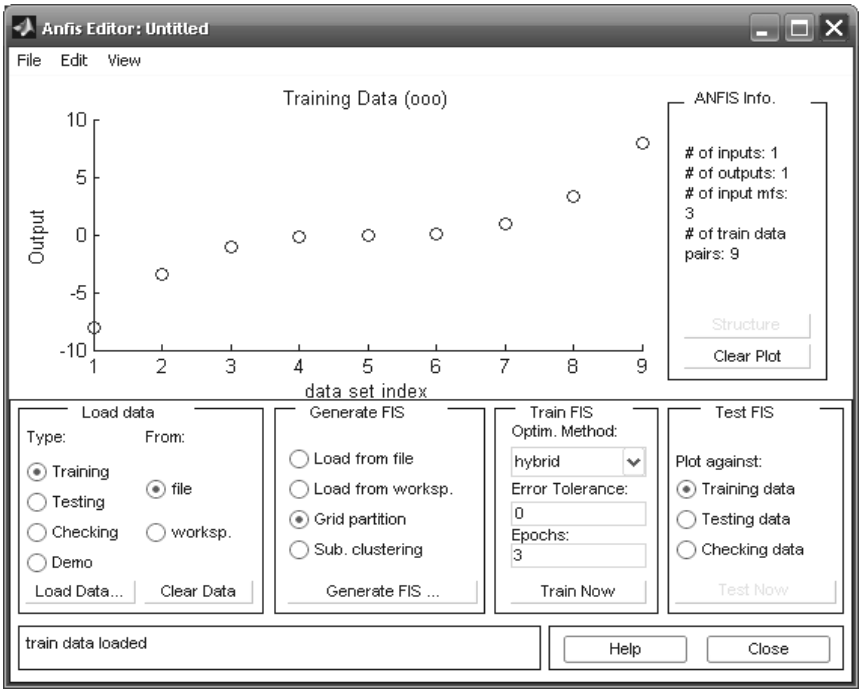


Рисунок 1.3 – Вікно редактора гібридних систем

Графічний інтерфейс програми кластеризації

Програма *Clustering* (кластеризація) дозволяє виявляти центри кластерів, тобто точки в багатовимірному просторі даних, біля яких групуються (скупчуються) експериментальні дані.

Запуск програми *Clustering* виконується командою *findcluster*. На рисунку 1.4 наведено приклад використання програми.

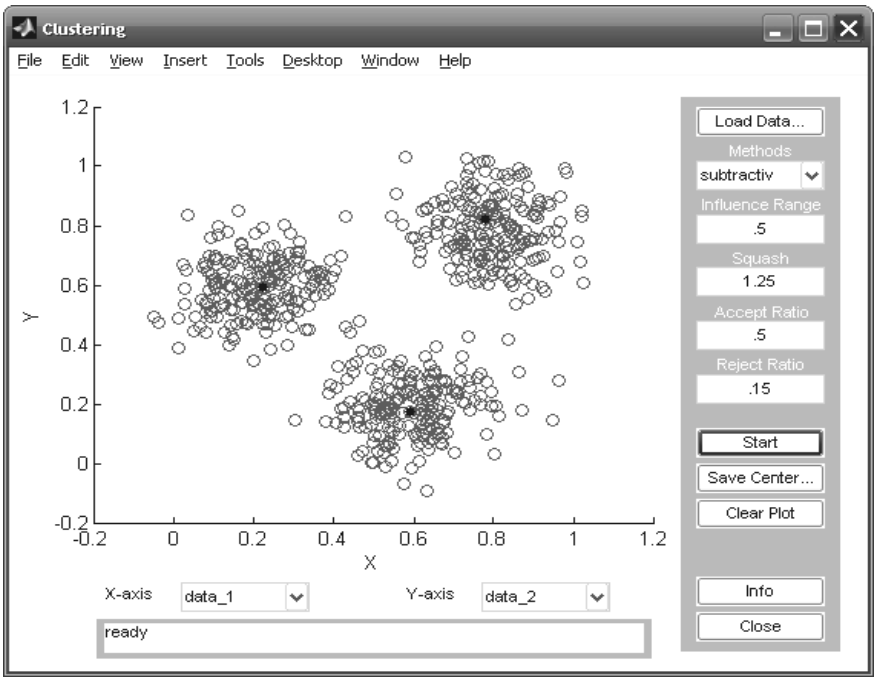


Рисунок 1.4 – Результат роботи програми *Clustering*

Робота в редакторі нечіткої системи виведення Fuzzy Inference System Editor

Для завантаження основного *fis-редактора* надрукуємо слово *fuzzy* в командному рядку. Після цього відкриється нове графічне вікно, зображене на рисунку 1.2. Для того щоб додати нову вхідну змінну, необхідно в меню *Edit* вибрати команду *Add Variable…\Input*. Для зміни імені змінної необхідно ввести нове ім'я в полі *Name* і натиснути клавішу *Enter*. Для того щоб задати ім'я системі, необхідно в меню *File* вибрати в підменю *Export* команду *To File* і ввести ім'я файла.

Щоб перейти в редактор функцій приналежності, необхідно двічі натиснути на будь-якій з функцій, де можна вибирати властивості конкретної, вікно відображено на рисунку 1.5:

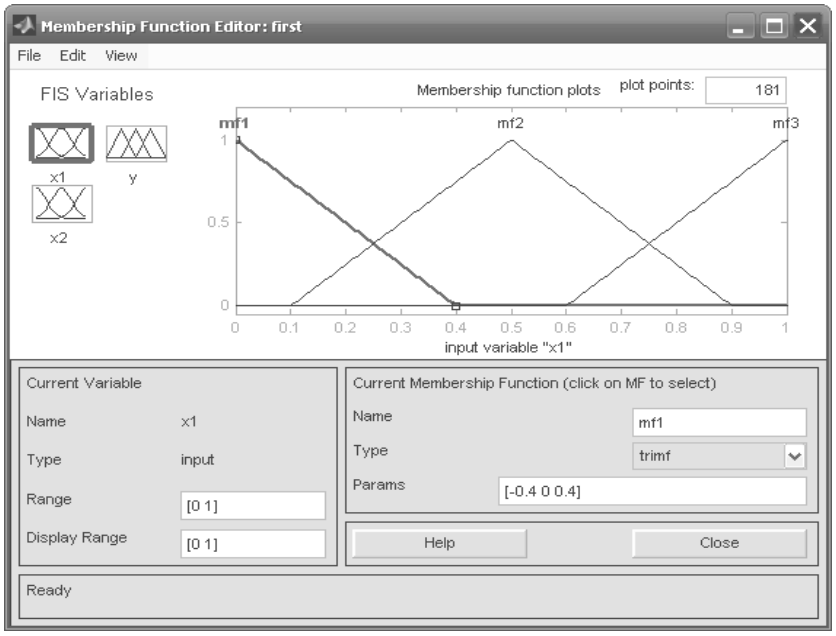


Рисунок 1.5 – Редактор функцій належності

Внизу вікна вказуються наступні властивості функції:

Current Variable:

- Name – ім'я функції;
- Type – тип (вхідна чи вихідна);
- Range – діапазон змінної;
- Display Range – відображуваний діапазон.

Current Membership Function:

– Name – ім’я поточної функції належності;

– Type – тип терму функції належності – вибирається з переліку (трикутна, трапецеїдальна, гауссові 1 та 2-го порядку та інші);

– Params – числові значення терму функції належності.

Для задання нових функцій належності для змінної необхідно в меню *Edit* вибрати команду *Add MFs*... У результаті з'явитися діалогове вікно (рисунок 1.6) вибору типу і кількості функцій належності.

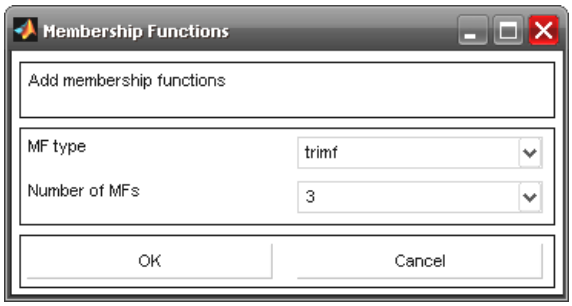


Рисунок 1.6 – Задання функцій належності

Вибравши необхідну кількість термів та їх тип, натиснути OK.

Ім’я та числове значення термів можна змінити, виділивши необхідний, і задати у відповідних полях області *Current Membership Function* нові значення.

Редактор бази знань RuleEditor

Для виклику редактора необхідно вибрати в меню *Edit* команду *Rules*..., відобразиться головне вікно, зображене на рисунку 1.7.

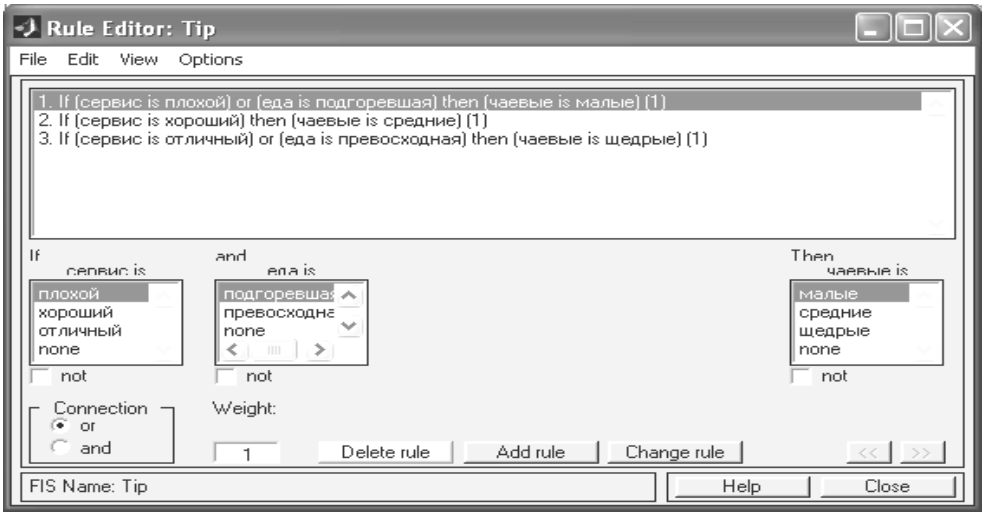


Рисунок 1.7 – Редактор правил

Для створення нових правил необхідно вибрати відповідну комбінацію термів і залежностей, вибрати тип зв'язку: *or* або *and*, вагу правила *Weight*, значення вихідної змінної та натиснути кнопку *Add rule*.

Для перегляду вікна візуалізації нечіткого логічного виведення викликаємо його командою *View rules*... меню *View*.

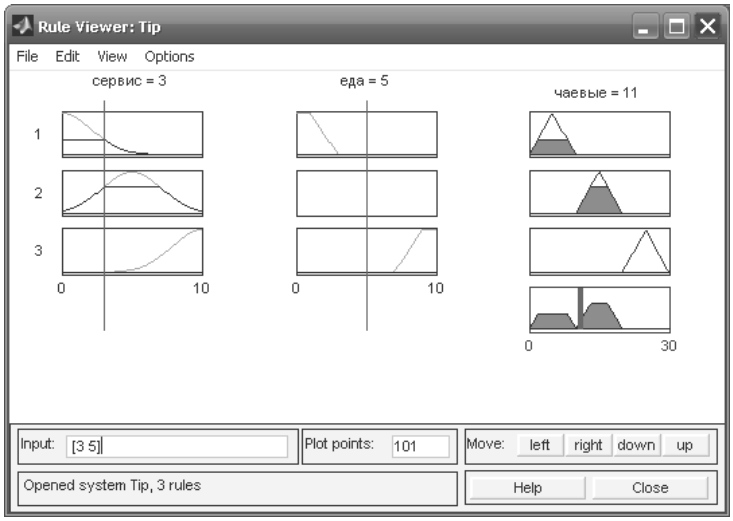


Рисунок 1.8 – Візуалізація нечіткого логічного виведення в *RuleViewer*

Можна переглянути поверхню "входи-виход", відповідну синтезованій нечіткій системі. Для виведення цього вікна необхідно використовувати команду *View surface***...** меню *View*.

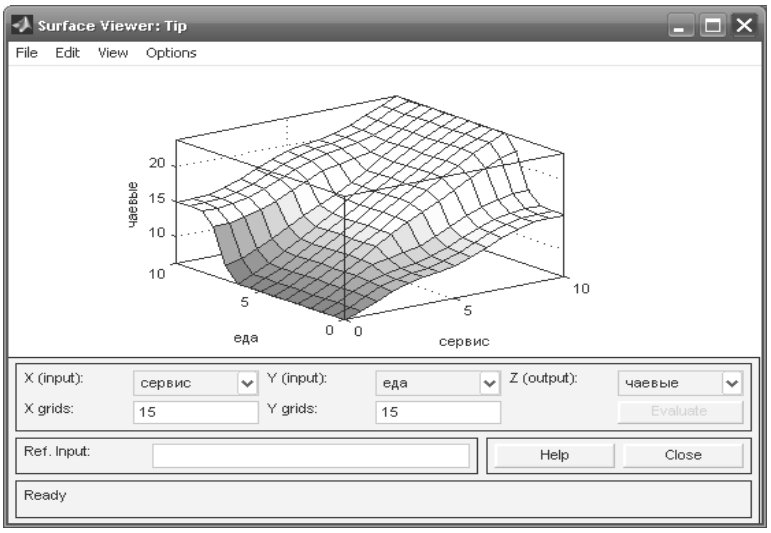


Рисунок 1.9 – Поверхня відгуку нечіткої системи

Порядок виконання роботи

1. Реалізувати експертну систему, що керує кутом повороту крана гарячої води, для підтримання напору та температури на необхідному рівні.

2. Відкрити редактор нечіткої системи виведення, прописавши в режимі командного рядка *fuzzy*.

У *Fuzzy Logic* – редакторі створити нечітку експертну систему, що має дві вхідні змінні, а саме: температура (назва змінної в редакторі *temp*) та напір води (назва змінної – *head*). Вихідна змінна – це вихід, що генерує система на базі експертних правил. Для даної системи вихідна змінна – це кут повороту крана гарячої води (назва – *valve*). Створити всі змінні та дати їм відповідні імена.

Зберегти створену систему з ім'ям *Control_temp*, вибравши в меню *File->Export->To File…*.

3. Перейти в редактор функцій належності, натиснувши двічі на будь-якій з функцій. Для вхідних та вихідної змінної необхідно визначити терми, а саме їх кількість, вид (трикутні, трапецеїдальні, ґауссові та ін.), діапазон зміни.

4. Візьмемо для вхідної змінної *temp* три терми: холодна (*cold*), середня (*mid*), гаряча (*hot*). Діапазон для температури [10; 80]. Задайте в редакторі відповідні значення для термів вхідної змінної *temp*, вибравши тип термів трикутний: *cold* [10 20 35], *mid* [30 35 40], *hot* [40 50 80].

5. Вхідна змінна *head* характеризує напір води, візьмемо наступні терми: малий (*small*), нормальний (*norm*) та великий (*big*). Діапазон зміни параметра [0;1] у відносних одиницях. Можна задати наступні числові значення термів: *small* [0 0.1 0.3], *norm* [0.25 0.5 0.75], *big* [0.6 0.8 1].

6. Вихідна змінна *valve* характеризує, наскільки необхідно повернути кран гарячої води і в який бік – на закриття чи відкриття. Для більш точного регулювання визначимо п'ять термів: відкрити швидко (*open_q*), відкрити повільно (*open_s*), не змінювати (*norm*), закрити повільно (*close_s*), закрити швидко (*close_q*). Значення для термів узяти у відносних одиницях у діапазоні [-10;10]. Числові значення для термів: *open_q* [-10 -7 -5], *open_s* [-6 -3 -1], *norm* [-2 0 2], *close_s* [1 3 6], *close_q* [5 7 10].

7. Після визначення всіх змінних необхідно створити базу знань, що може складатися з будь-яких правил, але чим точніше сформулювати правила, тим краще буде працювати система. Відкрити редактор правил *Rule Editor* та задати в ньому наступні правила для експертної системи:

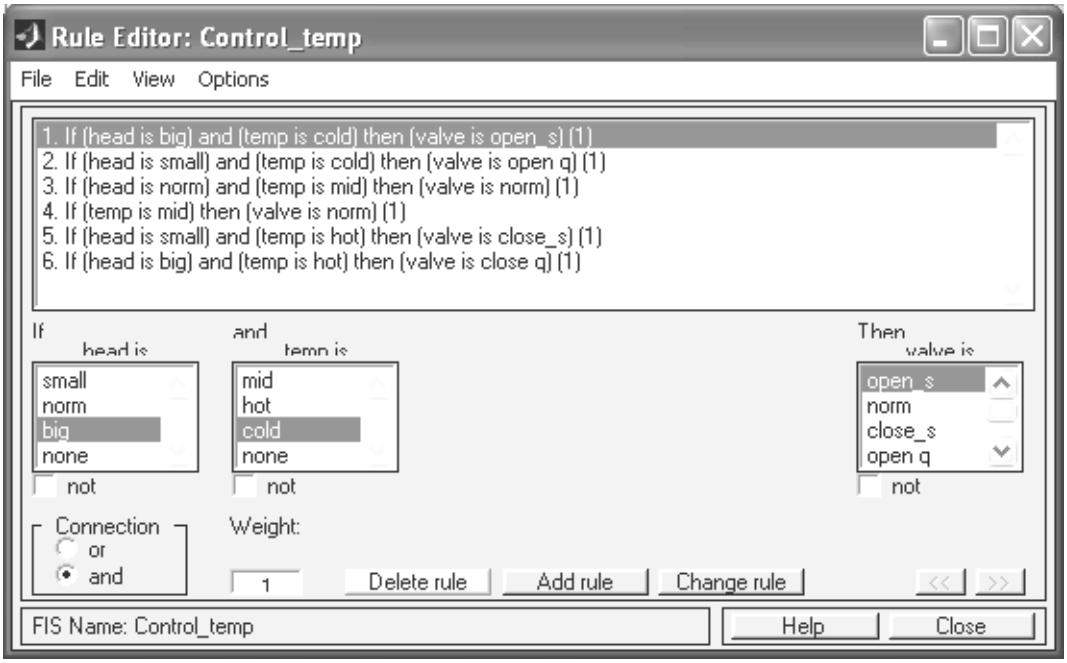


Рисунок 1.10 – Вікно редактора правил

8. Перевірити роботу системи, задаючи у вікні *Rule Viewer* значення для вхідних змінних (рис. 1.11). А також переглянути поверхню відгуку системи, вибравши з меню *View->Surface*.

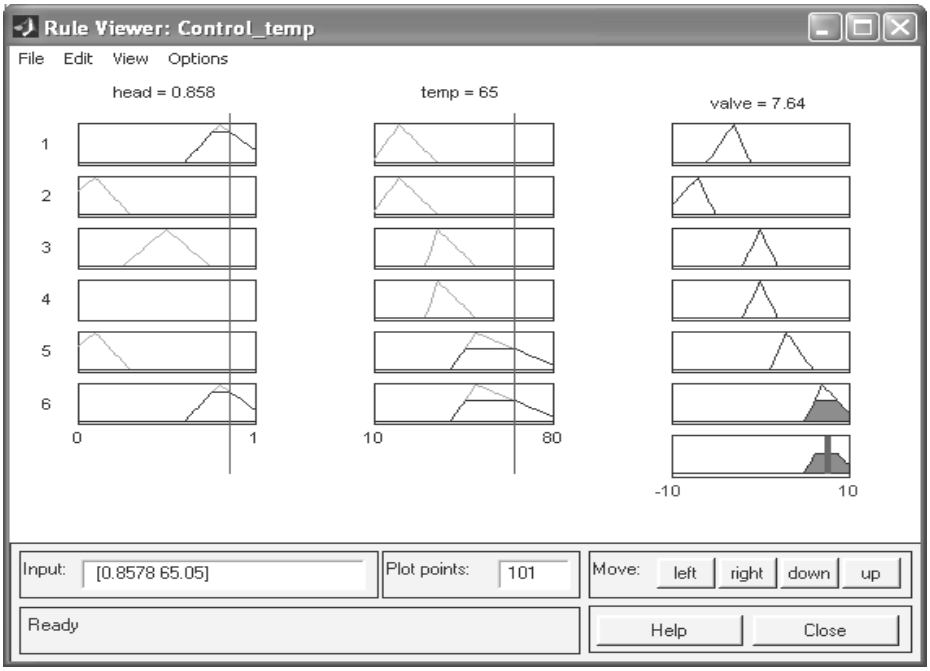


Рисунок 1.11 – Вікно перегляду роботи правил

9. Для розробленої системи самостійно змінити параметри функцій належності для досягнення кращих результатів роботи. Наприклад, можна змінити числові значення термів, тип функцій належності (трапецеїдальні, гауссові 1 та 2-го порядку), додати правила в базу знань та ін.

10. Порівняти отримані результати, зробити висновки з роботи.

Зміст звіту

1. Указати номер, тему й мету лабораторної роботи.

2. Навести *fis – структуру* експертної системи.

3. Відобразити початкові та оптимізовані функції належності та базу правил.

4. Зробити порівняльні висновки стосовно роботи системи з різним настройками.

Контрольні питання

1. Для чого призначений пакет *Fuzzy Logic Toolbox*? Назвати основні властивості.

2. Для чого призначений графічний редактор нечіткої системи виведення?

3. Як викликати *FIS–редактор*?

4. Для чого призначений графічний інтерфейс гібридних мереж?

5. Що дозволяє виконувати програма кластеризації?

6. Дати визначення експертної системи.

7. Що таке функції належності? Як викликати редактор для їх задання?

8. Які види функцій належності ви знаєте?

9. Де можна задати правила для системи?

10. Що таке поверхня відгуку нечіткої системи? Як її переглянути?

11. Які галузі застосування подібних експертних систем?

12. Пояснити принцип дії даної експертної системи.

Література: [9–10, 12].

Лабораторна робота № 2

Тема. Порівняльне дослідження роботи алгоритмів Мамдані та Сугено в однотипних додатках

Мета: освоїти принципи роботи алгоритмів Мамдані та Сугено. Навчитися задавати властивості системи залежно від поставленої задачі та алгоритму розв'язання.

Короткі теоретичні відомості

Алгоритми нечіткого виведення різняться, головним чином, видом використовуваних правил, логічних операцій і різновидом методу дефазифікації. Розроблені моделі нечіткого виведення Мамдані, Сугено, Ларсена, Цукамото. При розгляді алгоритмів для спрощення припустимо, що базу знань організують два нечітких правила вигляду:

$П_1$: якщо x є A_1 та y є B_1, то z є C_1,

$П_2$: якщо x є A_2 та y є B_2, то z є C_2,

де x і y – імена вхідних змінних, z – ім'я змінної виведення, A_1, A_2, B_1, B_2, C_1, C_2 – деякі задані функції належності, при цьому чітке значення z_0 необхідно визначити на основі наведеної інформації та чітких значень x_0 і y_0.

Алгоритм Мамдані (Mamdani)

Алгоритм Мамдані є одним з перших, який знайшов застосування в системах нечіткого виведення. Він був запропонований 1975 р. англійським математиком Е. Мамдані (Ebrahim Mamdani) як метод для керування паровим двигуном. Формально *алгоритм Мамдані* може бути визначений таким чином.

1. Процедура фазифікації: визначаються ступені істинності, тобто значення функцій належності для лівих частин кожного правила (передумов): $A_1(x_0)$, $A_2(x_0)$, $B_1(y_0)$, $B_2(y_0)$.

2. Нечітке виведення: знаходяться рівні відтинання для передумов кожного з правил з використанням операції мінімум:

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),$$

$$\alpha_2 = A_2(x_0) \wedge B_2(y_0),$$

де через «$\wedge$» позначена операція логічного мінімуму (min), потім знаходяться

«зрізані» функції належності

$$C_1'(z) = (\alpha_1 \wedge C_1(z)),$$

$$C_2'(z) = (\alpha_2 \wedge C_2(z)).$$

3. Композиція: з використанням операції максимуму (max, позначення: «∨») виконується об'єднання знайдених зрізаних функцій, що приводить до отримання підсумкової нечіткої підмножини для змінної виходу з функцією належності

$$\mu_\Sigma(z) = C(z) = C_1'(z) \vee C_2'(z) = (\alpha_1 \wedge C_1(z)) \vee (\alpha_2 \wedge C_2(z)).$$

4. Приведення до чіткості (для знаходження z_0) проводиться, наприклад, центроїдним методом (як x – координата центра ваги функції належності підсумкової нечіткої підмножини для змінної виходу):

$$z_0 = \frac{\int_\Omega z \cdot \mu_\Sigma(z) dz}{\int_\Omega \mu_\Sigma(z) dz}$$

Алгоритм ілюструється рис. 2.1:

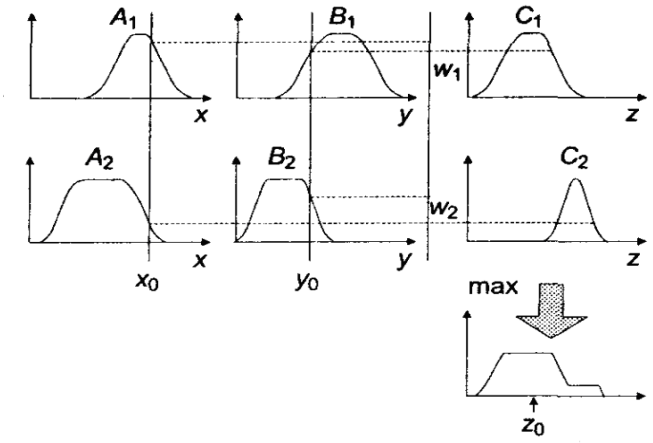


Рисунок 2.1 – Графічна реалізація

Алгоритм Сугено (Sugeno)

Формально алгоритм Сугено, запропонований Сугено і Такагі, може бути визначений таким чином.

1. Перший етап – як в алгоритмі Мамдані.
2. На другому етапі знаходяться $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$, $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$ та індивідуальні виходи правил:

$z^*_1 = a_1x_0 + b_1y_0$,

$z^*_2 = a_2x_0 + b_2y_0$,

3. На третьому етапі визначається чітке значення змінної виведення:

$$z_0 = \frac{\alpha_1 z_1^* + \alpha_2 z_2^*}{\alpha_1 + \alpha_2}$$

Алгоритм ілюструється на рисунку 2.2:

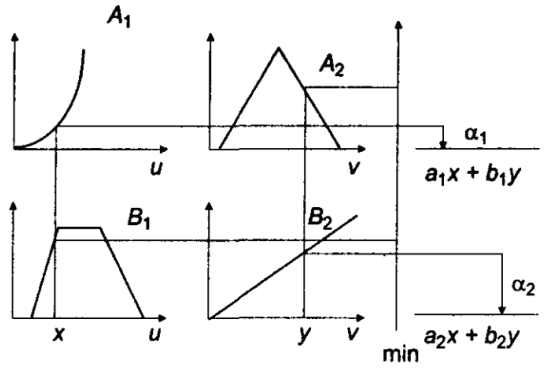


Рисунок 2.2 – Графічна реалізація алгоритму Сугено

Порядок виконання роботи

1. Порівняти алгоритми Мамдані і Сугено на прикладі створення системи нечіткого логічного виведення, що моделює залежність $y = x_1^2 \cdot \sin(x_2 - 1)$, $x_1 \in [-7,\ 3]$, $x_2 \in [-4.4,\ 1.7]$. Проектування системи нечіткого логічного виведення необхідно провести на основі графічного зображення вказаної залежності.

2. Для побудови тривимірного зображення функції $y = x_1^2 \cdot \sin(x_2 - 1)$ в межах $x_1 \in [-7,\ 3]$, $x_2 \in [-4.4,\ 1.7]$ необхідно скласти наступну програму, прописавши її в m-файлі:

%Побудова графіка функції y = x1^2*sin(x2-1) в межах x1є[-7, 3] і x2є[-4.4, 1.7].

```
n = 15;                          % кількість точок
x1 = -7:10/(n-1):3;              % задання параметрів змінної x1
x2 = -4.4:6.1/(n-1):1.7;         % задання параметрів змінної x1
y = zeros (n, n);                % формування нульового масиву
                                 % розміром n×n для вихідної змінної
for j = 1:n
```

```
y (j,:) = x1.^2*sin(x2(j)-1);
end
surf (x1, x2, y)                    % зображення поверхні функції
xlabel ('x1')
ylabel ('x2')
zlabel ('y')
title ('Target');
```

У результаті виконання програми отримаємо графічне зображення, наведене на рис. 2.3.

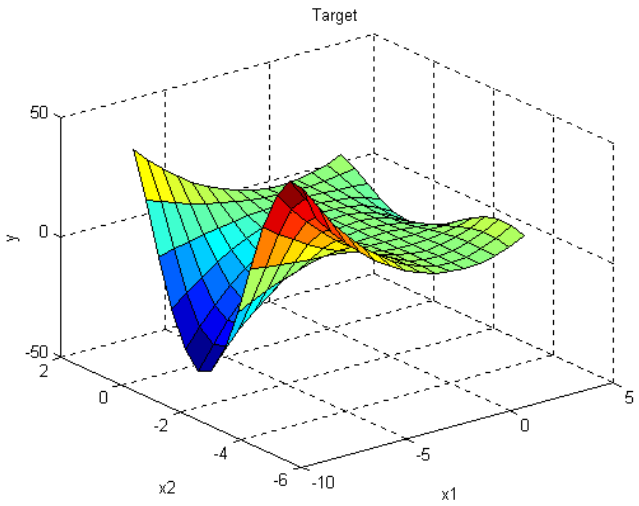


Рисунок 2.3 – Еталонна поверхня

3. Реалізуйте дві нечіткі системи для заданої функції, вибравши для першої тип системи Сугено, для другої – Мамдані.

4. Для створення першої системи завантажте fis-редактор. Виберіть тип системи – Sugeno. Додайте другу вхідну змінну та назвіть усі змінні відповідними іменами, а саме першу вхідну змінну перейменуйте на x1, другу – на x2, а вихідну – на y.

5. Перейдіть у редактор функцій належності. Задайте діапазон змінення змінної x1 та створіть для неї функції належності, вказавши ім’я, тип та числові значення термів:

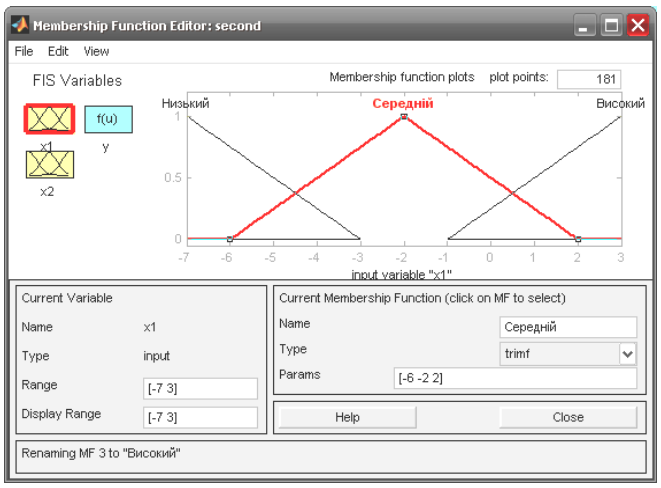


Рисунок 2.4 – Функції належності змінної x1

6. Аналогічно задайте діапазон змінення змінної x2 та створіть для неї функції належності, вказавши ім’я, тип та числові значення термів:

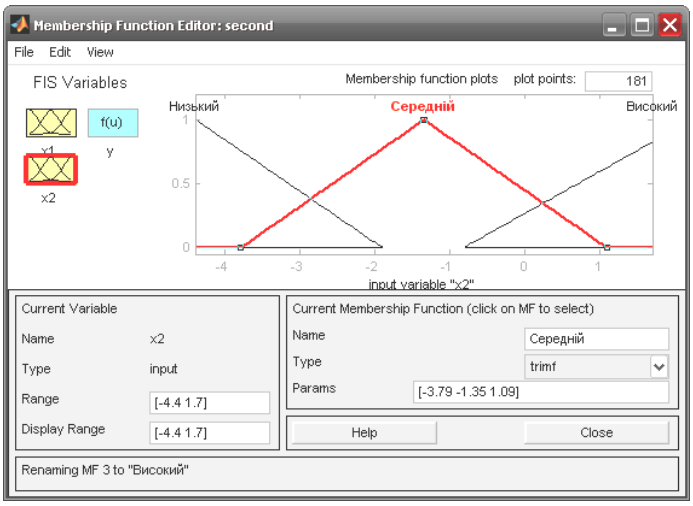


Рисунок 2.5 – Функції приналежності змінної x2

7. Для алгоритму Сугено для вихідної змінної задаються лінійні залежності між входами і виходом, що мають міститися в базі знань. У базі знань вказано 5 різних залежностей: y=50; y=4x1-x2; y=2x1+2x2+1; y=8x1+2x2+8; y=0. Тому додайте ще дві залежності шляхом вибору команди *Add Mfs* меню *Edit*. У діалоговому вікні в полі *Number of MFs* виберіть 2 і натисніть кнопку OK. Задайте найменування і параметри лінійних залежностей. Для цього виберіть першу залежність mf1. Надрукуйте найменування залежності, наприклад 50, у полі *Name* і встановіть тип залежності - константа шляхом вибору опції Constant в меню *Type*. Після цього введіть значення параметра 50 у полі *Params*. Аналогічно для другої залежності mf2 введіть найменування залежності, наприклад 8+8x1+2x2.

Потім вкажіть лінійний тип залежності шляхом вибору опції Linear у меню *Type* і введіть параметри залежності 8 2 8 в полі *Params*. Для лінійної залежності порядок параметрів наступний: перший параметр – коефіцієнт при першій змінній, другий, – при другій і т. д., останній параметр – вільний член залежності. У результаті маєте отримати графічне вікно, зображене на рис. 2.6.

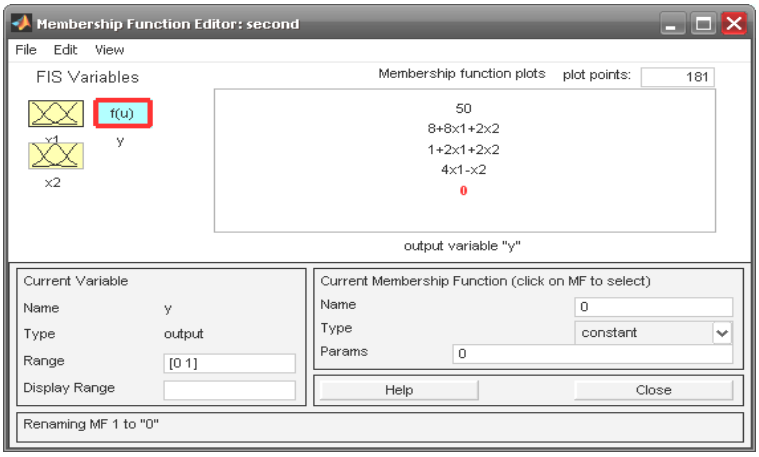


Рисунок 2.6 – Вікно лінійних залежностей “входи-вихід”

8. Аналізуючи еталонну поверхню, можна скласти наступні залежності та правила:

якщо x1=середній, то y=0;

якщо x1=високий і x2=високий, то y=2x1+2x2+1;

якщо x1=високий і x2=низький, то y=4x1-x2;

якщо x1=низький і x2=середний, то y=8x1+2x2+8;

якщо x1=низький і x2=низький, то y=50;

якщо x1=низький і x2=високий, то y=50.

9. Перейдіть у редактор бази знань *RuleEditor* і введіть правила бази знань, що наведені вище.

10. Перегляньте вікно візуалізації нечіткого логічного виведення, а також поверхню “входи–вихід” для синтезованої нечіткої системи.

11. Реалізуйте нечітку логічну систему для заданої функції, використовуючи алгоритм Мамдані. Самостійно складіть правила для відповідних функцій.

12. Порівняйте отримані різними методами поверхні з еталонною поверхнею. Зробіть висновки щодо ефективності кожної з них.

Зміст звіту

1. Указати номер, тему й мету лабораторної роботи.

2. Зобразити FIS-структури для розроблених систем за різними алгоритмами.

3. Навести перелік правил.

4. Відобразити отримані результати – перехідні процеси, поверхні відгуку.

5. Зробити порівняльні висновки стосовно роботи системи з різним настройками.

Контрольні питання

1. Наведіть алгоритми нечіткого виведення, які ви знаєте.

2. Перелічіть методи дефазифікації.

3. Який метод дефазифікації використовується в алгоритмі Мамдані?

4. Як створити нечітку систему за алгоритмом Сугено?

5. Що таке функції належності? Які типи функцій належності ви знаєте?

6. Як задаються функції належності для різних систем?

Література: [6–9].

Лабораторна робота № 3

Тема. Створення інтелектуальної нейронної мережі для задач апроксимації та для передбачення значення функції за допомогою пакета Neural Networks

Мета: ознайомитися з пакетом прикладних програм Neural Networks. Вивчити принципи побудови нейронної мережі, користуючись набором функцій пакета та за допомогою елементів графічного інтерфейсу.

Короткі теоретичні відомості

Пакет *Neural Networks Toolbox* (нейронні мережі) містить засоби для проектування, моделювання, навчання і використання безлічі відомих парадигм апарата штучних нейронних мереж (НМ): від базових моделей персептрона до найсучасніших асоціативних мереж, що самоорганізуються. Пакет може бути використаний для розв'язання безлічі всіляких завдань, таких як обробка сигналів, нелінійне керування, фінансове моделювання і подібне.

До пакета включено більше 15 відомих типів мереж і навчальних правил, що дозволяють користувачеві вибирати найбільш відповідну для конкретного застосування або дослідного завдання парадигму. Для кожного типу архітектури і навчальних правил є функції ініціалізації, навчання, адаптації, створення і моделювання, демонстрації та приклад додатка мережі.

За допомогою нейронних мереж розв'язують ряд задач.

Класифікація образів. Задача полягає у зазначенні належності вхідного образу, що представлений вектором ознак, до одного чи до декількох попередньо визначених класів. До відомих додатків належать розпізнавання букв, розпізнавання мови, класифікація сигналу електрокардіограми, класифікація клітин крові та ін.

Кластеризація / категоризація. При розв'язанні задачі кластеризації, яка відома як класифікація образів «без вчителя», відсутня навчальна вибірка з позначками класів. Алгоритм кластеризації оснований на подібності образів і переносить близькі образи в один кластер.

Апроксимація функцій. Припустимо, що є навчальна вибірка *((x[1], y[1]), (x[2], y[2]), ..., (x[N], y[N]))* (пари даних «вхід–вихід»), яка генерується невідомою функцією *F(x)*, що зашумлена. Задача апроксимації полягає в знаходженні оцінки невідомої функції *F(x)*. Апроксимація функцій необхідна при розв'язанні багаточисельних інженерних і наукових задач моделювання.

Передбачення / прогноз. Нехай задано *n* дискретних відгуків *{y(t1), y(t2), ..., y(tk)}* у послідовні моменти часу $t_1, t_2, ..., t_k$. Задача полягає в передбаченні значення $y(t_{k+1})$ у деякий майбутній момент часу t_{k+1}. Передбачення / прогноз чинять значний вплив на прийняття рішень у бізнесі, науці і техніці.

Оптимізація. Багаточисельні проблеми в математиці, техніці, науці, медицині та статистиці можуть розглядатися як проблеми оптимізації. Задачею алгоритму оптимізації є знаходження такого розв'язку, який задовольняє систему обмежень і максимізує або мінімізує цільову функцію.

Керування. Розглянемо динамічну систему, задану сукупністю *{u(t), y(t)}*, де *u(t)* є вхідним керуючим впливом, а *y(t)* – виходом системи в момент часу *t*. У системах керування з еталонною моделлю метою керування є розрахунок такого вхідного впливу *u(t)*, при якому система рухається по бажаній траєкторії, що диктується еталонною моделлю.

Огляд пакета прикладних програм Neural Networks Toolbox

Виклик пакета здійснюється командою *nntool* з командного рядка MATLAB або з вікна запуску додатків з розділу Neural Network Toolbox.

Після запуску пакета відкривається вікно *Network/Data Manager* – керування мережею та даними (рис. 3.1).

Призначення кнопок у менеджері наступне:

– *Help* – кнопка виклику вікна підказки *Network/Data Manager Help*;

– *New Data...* – кнопка виклику вікна формування даних *Create New Data* (рис. 3.2);

– *New Network...* – кнопка виклику вікна створення нової нейронної мережі *Create New Network* (рис. 3.3);

– *Import...* – кнопка виклику вікна для імпорту або завантаження даних *Import or Load to Network/Data Manager*;

– *Export...* – кнопка виклику вікна для експорту або запису даних у файл *Export or Save from Network/Data Manager*.

Щоб створити нейронну мережу, необхідно виконати наступні операції:

– сформувати послідовності входів і цілей (кнопка *New Data*) або завантажити їх з робочої області системи MATLAB чи з файла (кнопка *Import*);

– створити нову нейронну мережу (кнопка *New Network*) або завантажити її з робочої області системи MATLAB чи з файла (кнопка *Import*);

– вибрати тип нейронної мережі та натиснути кнопку *Train...*, щоб відкрити вікно для задання параметрів процедури навчання;

– відкрити вікно *Network* для перегляду, ініціалізації, моделювання, навчання й адаптації мережі.

Рисунок 3.1 – Вікно *Network/Data Manager*

Вікно Create New Data

Вікно *Create New Data* (рис. 3.2) включає 2 ділянки редагування тексту для запису імені даних, що вводяться (ділянка Name), і введення самих даних

(ділянка Value), а також 6 кнопок для вказівки типу даних, що вводяться.

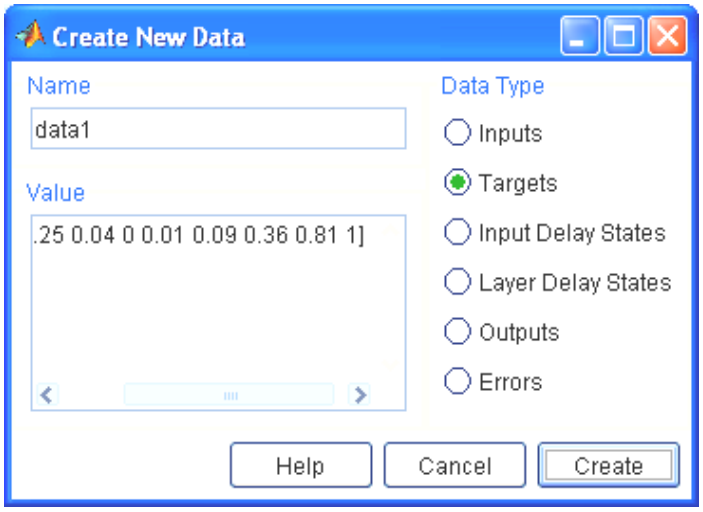


Рисунок 3.2 – Вікно *Create New Data*

Розрізняють наступні типи даних:

Inputs (Входи) – послідовність значень входів;

Targets (Цілі) – послідовність значень цілей;

Input Delay States (Стан лінії затримки входу) – початкові умови лінії затримки на вході;

Layer Delay States (Стан лінії затримки шару) – початкові умови лінії затримки в шарі;

Outputs (Виходи) – послідовність значень виходу мережі;

Errors (Помилки) – різниця значень цілей і виходів.

Вікно Create New Network

Вікно *Create New Network* (рис. 3.3) включає поля для задання параметрів створюваної мережі. Залежно від типу мережі кількість полів і їх назв змінюються.

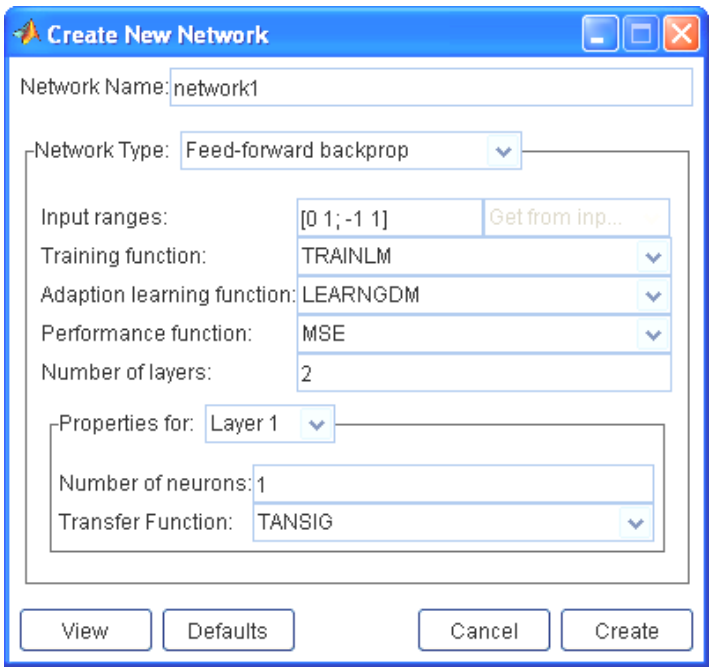


Рисунок 3.3 – Вікно *Create New Network*

У вікні *Create New Network* розташовані наступні поля:

– *Network Name* (Ім'я мережі) – стандартне ім'я мережі, що присвоюється GUI-інтерфейсом NNTool;

– *Network Type* (Тип мережі) – список мереж, доступних для роботи з інтерфейсом NNTool;

– *Input ranges* (Діапазони входу) – припустимі межі входів, які або призначаються користувачем, або визначаються автоматично за ім’ям вхідної послідовності, що вибирається зі списку *Get from Inp...*;

– *Training function* (Функція навчання) – список навчальних функцій;

– *Adaption learning function* (Функції настроювання для режиму адаптації) – список функцій настроювань;

– *Performance function* (Функція якості навчання) – список функцій оцінки якості навчання;

– *Number of layers* (Кількість шарів) – кількість шарів нейронної мережі;

– *Properties for* (Властивості) – список шарів: Layer 1 (Шар 1), Layer 2 (Шар 2);

– *Number of neurons* (Кількість нейронів) – кількість нейронів у шарі;

– *Transfer function* (Функція активації) – функція активації шару.

Діалогова панель Network

Діалогова панель *Network* (рис. 3.4) відкривається тільки в тому випадку, коли у вікні *Network/Data Manager* виділена створена мережа й потім була натиснута одна із кнопок *View*, *Initialize*, *Simulate*, *Train*, *Adapt*.

Панель має 6 закладок:

– *View* (Перегляд) – структура мережі;
– *Initialize* (Ініціалізація) – задання початкових ваг і зсувів;
– *Simulate* (Моделювання) – моделювання мережі;
– *Train* (Навчання) – навчання мережі;
– *Adapt* (Адаптація) – адаптація й настроювання параметрів мережі;
– *Weights* (Ваги) – перегляд установлених ваг і зсувів.

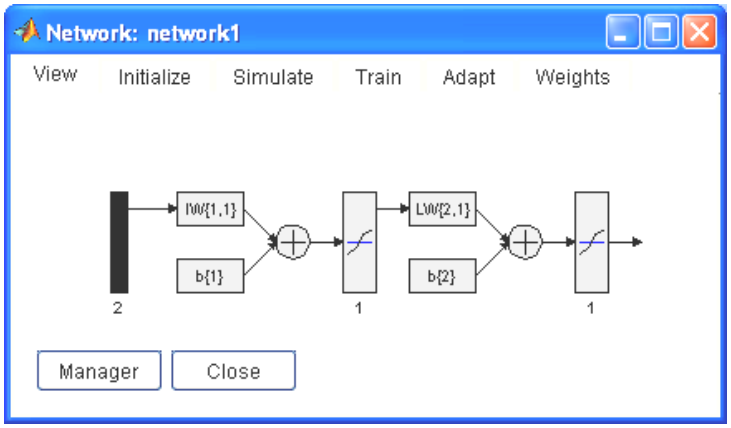


Рисунок 3.4 – Діалогова панель *Network*

Огляд функцій пакета Neural Networks Toolbox

До складу пакета Neural Networks входять більше 150 різних функцій, утворюючи собою своєрідну макромову програмування і дозволяючи користувачеві створювати, навчати і використовувати різноманітні НМ. Дані функції за своїм призначенням діляться на ряд груп:

– функції активації (передавальні функції) і пов'язані з ними функції;
– функції навчання нейронних мереж;
– функції настроювання шарів нейронів;
– функції одновимірної оптимізації;
– функції ініціалізації шарів і зсувів;

- функції створення нейронних мереж;
- функції перетворення входів мереж;
- функції ваг і відстаней;
- функції розміщення нейронів (топологічні функції);
- функції використання нейронних мереж;
- графічні функції та ін.

Порядок виконання роботи

1. Створіть узагальнено-регресійну нейронну мережу (НМ) з ім'ям *a* для апроксимації функції вигляду $y=x^2$ на відрізку [-1, 1], використовуючи наступні експериментальні дані:

x = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1],

y = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1].

Процедура створення і використання даної НМ описується таким чином:

```
% Задання вхідних значень
P = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1];
% Задання вихідних значень
T = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1];
% Створення узагальнено-регресійної НМ з відхиленням 0,01
a = newgrnn (P, T, 0.01);
% Опитування НМ
Y = sim (a, [-0.9 -0.7 -0.3 0.4 0.8])
```

2. Визначте точність апроксимації з отриманих даних. Спробуйте поліпшити якість апроксимації, реалізувавши мережу з радіальними базисними елементами, використовуючи функцію *newrb.*

3. Порівняйте роботу різних мереж для завдань апроксимації.

4. Запустіть графічний редактор NNTool та створіть для тих самих вхідних даних та цільової функції нейронну радіально-базисну мережу з нульовою помилкою (Radial basis (exact fit)) та радіально-базисну мережу з мінімальним числом нейронів (Radial basis (fewer neurons)).

5. Виконавши експорт (Export…) розроблених мереж до робочої області, перевірте правильність їх роботи, задавши в командному рядку MatLab дані для опитування.

6. Порівняйте результати роботи мереж, розроблених програмно та за допомогою графічного інтерфейсу NNTool.

7. Отримайте графіки цільових функцій та результати, що видала мережа. Зробіть порівняльний аналіз усіх отриманих графіків.

8. Побудуйте лінійну нейронну мережу, що дозволяє прогнозувати майбутнє значення сигналу (функції часу), що описується співвідношенням $x(t)=cos(4\pi t)$, який піддається дискретизації з інтервалом 0,01с. Використовувати нижченаведений лістинг.

```
t=0:0.01:5
x=cos(t*4*pi)
plot(t,x)
q=length(x)
p=zeros(5,q)
p(1,2:q)=x(1,1:(q-1))
p(2,3:q)=x(1,1:(q-2))
p(3,4:q)=x(1,1:(q-3))
p(4,5:q)=x(1,1:(q-4))
p(5,6:q)=x(1,1:(q-5))
s=newlind(p,x)
y=sim(s,p)
plot(t,y,t,x,'*')
e=x-y
plot(t,e)
```

9. Наведіть отримані графіки для заданої функції.

10. Відобразьте графік отриманої помилки прогнозу.

11. Оцініть точність прогнозу і зробіть висновки про роботу НМ.

Зміст звіту

1. Указати номер, тему й мету лабораторної роботи.

2. Відобразити реалізацію мережі з радіальними базисними елементами для задачі апроксимації.

3. Зробити висновки, щодо роботи різних мереж для задачі апроксимації.

4. Надати програмний код зі створення лінійної нейронної мережі для прогнозування значення функції.

5. Відобразити отримані графіки: вихідний сигнал, прогноз мережі та похибку прогнозу.

6. Зробити висновок щодо точності прогнозу для заданого сигналу.

Контрольні питання

1. Що собою являє нейронна мережа?

2. Які проблеми розв’язують у контексті нейронних мереж?

3. Поясніть структуру і властивості штучного нейрона.

4. Перелічіть функції активації нейронів.

5. Які нейронні мережі виділяють за топологією?

6. Як навчаються нейронні мережі?

7. Які сфери застосування нейронних мереж?

8. Назвіть можливості пакета Neural Networks.

9. Які різновиди нейронних мереж ви знаєте?

Література: [7–9, 12].

Лабораторна робота № 4

Тема. Моделювання багатошарової нейронної мережі

Мета: навчитися проектувати та реалізовувати нейронну мережу для задач апроксимації з урахуванням шуму.

Короткі теоретичні відомості

При проектуванні нейронної мережі необхідно визначити кількість шарів і кількість елементів (нейронів) у кожному шарі. Вибір архітектури нейронної мережі залежить від складності завдання і ґрунтується на досвіді розробника. При моделюванні нейронної мережі виділяють наступні етапи:

1) створення мережі;

2) навчання мережі;

3) тестування мережі;

4) використання мережі для розв'язання поставленого завдання.

Приклад розрахунку параметрів мережі.

При створенні двошарової нейронної мережі прямого розповсюдження необхідно визначити число нейронів у кожному шарі. На вхід мережі подається ідеальний (незашумлений) вектор p. Його компоненти розташовані на відрізку $[-1;1]$ з кроком $0{,}05$ (вибраним експериментально), тобто розмірність вхідного вектора $n = 41$.

Вихідний вектор t вибирають таким чином, щоб його компоненти відповідали векторам входу.

У нейропакеті NeuralNetworkToolbox системи Matlab задання вхідного і вихідного вектора записується наступним чином:

$p = [-1 : 0{,}05 : 1]$ – вхідний вектор

$t = \sin(2\pi \cdot p)$ – вихідний (цільовий) вектор

Число нейронів у прихованому шарі визначається за формулою:

$$\frac{mN}{1+\log_2 N} \le L_\omega \le m\left(\frac{N}{m}+1\right)(n+m+1)+m,$$

де n – число вхідних нейронів ($n=1$); m – число вихідних нейронів ($m=1$); N – число елементів навчальної вибірки ($N=41$); L_ω – число синаптичних ваг.

Підставивши відповідні значення, отримаємо

$$6 \le L_\omega \le 127 .$$

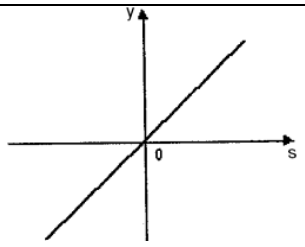
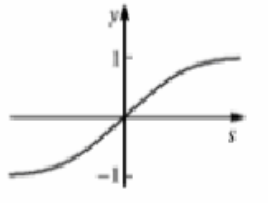
Оцінивши необхідне число ваг, розраховують число нейронів у прихованому шарі:

$$L = \frac{L_\omega}{n+m} \Rightarrow 3 \le L \le 62 .$$

Експериментально встановлено, що для задачі апроксимації кращі результати дає двошарова однонапрямлена мережа, що містить один нейрон у вхідному шарі, 30 нейронів в прихованому шарі і один нейрон у вихідному шарі.

Вихідний сигнал формального нейрона обчислює активаційна або передатна функція. Зазвичай це функція *purelin* або *tansig* (функції показані в таблиці 4.1). Зауважимо, що за замовчанням у пакеті NeuralNetworksToolbox використовується функція *tansig*.

Таблиця 4.1 – Види передавальних функцій

Передавальні функції	Графік	Формула	Діапазон значень
purelin		$f(s)=s$	$(-\infty;+\infty)$
tansig		$f(s)=\frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$	$(-1;1)$

Створювана багатошарова (двошарова) мережа – це мережа з безперервними сигналами, тому для обчислення вихідних сигналів прихованого шару використовується гладенька безперервна функція гіперболічного тангенса *tansig*.

Безперервність перших і других похідних дозволяє при навчанні мережі використовувати квазіньютонівський метод оптимізації алгоритмом зворотного поширення. Для обчислення вихідних сигналів вихідного шару використовується лінійна функція активації *purelin*.

Як навчальний алгоритм узято ітераційний градієнтний алгоритм зворотного поширення помилки, який використовується з метою мінімізації середньоквадратичного відхилення поточного і бажаного виходу нейронної мережі.

Для створення однонапрямленої нейронної мережі, що навчається із застосуванням алгоритму зворотного поширення в нейропакеті NeuralNetworksToolbox, використовується функція *newff*.

Синтаксис функції *newff* має вигляд:

net = newff (PR, [S_1 S_2 ... S_{Nl}], {TF_1 TF_2 ... TF_{Nl}}, BTF, BLF, PF)

Функція *newff* як вхідні параметри використовує:

– *PR* – матрицю мінімальних та максимальних значень для R вхідних елементів з розмірністю $R \times 2$. Для отримання матриці *PR* можна використовувати функцію *minmax*;

– S_i – кількість нейронів в *i*-му шарі, *N1* – кількість шарів;

– TF_i – функцію активації *i*-го шару, за замовчанням 'tansig';

– *BTF* – навчальну функцію зворотного поширення, за замовчанням 'trainlm';

– *BLF* – алгоритм підстроювання ваг і зміщень (навчальний алгоритм), за замовчанням 'learngdm';

– *PF* – функцію оцінки функціонування мережі, за замовчанням 'mse'.

Функція *newff* повертає однонапрямлену мережу, що складається з N шарів.

Для перегляду і подальшого моделювання створеної нейронної мережі застосовується функція *gensim (net),* яка генерує нейромережевий блок Simulink. Структурна схема мережі зображується за допомогою типового набору блоків, з'єднувальних елементів і позначень, прийнятих у пакеті NeuralNetworksToolbox і пакеті Simulink системи Matlab.

Для навчання створеної нейронної мережі необхідно вказати, які набори даних повинні бути використані як навчальні.

Після того як зібрано навчальний набір даних для проектованої мережі, проводиться автоматичне настроювання ваг і зміщень за допомогою процедури навчання, яка мінімізує різницю між бажаним сигналом і отриманим на виході в результаті моделювання мережі. Ця різниця має назву «помилки навчання». Використовуючи помилки навчання для всіх наявних спостережень, можна сформувати функцію помилок або критерій якості навчання. Таким критерієм може слугувати середньоквадратична помилка.

Алгоритм зворотного поширення помилки діє ітеративно, по кроках, які називають епохами або циклами. Величина кроку визначає швидкість навчання. При великому кроці є велика ймовірність пропуску абсолютного мінімуму, а при малому кроці зростає час навчання.

На кожному циклі на вхід мережі послідовно подаються всі навчальні спостереження, вихідні значення порівнюються з цільовими значеннями і обчислюється функція критерію якості навчання – функція помилки. Значення функції помилки, а також її градієнта використовуються для коригування ваг і зміщень, після чого всі дії повторюються. Процес навчання припиняється, якщо:

- реалізовано задану кількість циклів;
- помилка досягла заданої величини;
- помилка досягла деякого значення і перестала зменшуватися;
- градієнт помилки став менше заданого.

Порядок виконання роботи

1. Необхідно створити нейронну мережу та виконати апроксимацію функції $y = \sin(2\pi \cdot x)$ $x \in [-1; 1]$ з урахуванням впливу шуму.

2. Створіть пустий m-файл та збережіть його в робочому каталозі. Задайте вхідний вектор *p* і вихідний – *t*.

3. З урахуванням розрахованих параметрів, наведених вище, сформуйте модельовану мережу за допомогою команди:

net = newff (minmax(p), [30, 1], {'tansig', 'purelin'}, 'trainbfg');

4. Для реалізації нейронної мережі згенеруйте нейромережевий блок Simulink, задавши в m-файл наступну функцію:

gensim (net);

У результаті повинна відкритися структурна схема мережі, зображена на рис. 4.1.

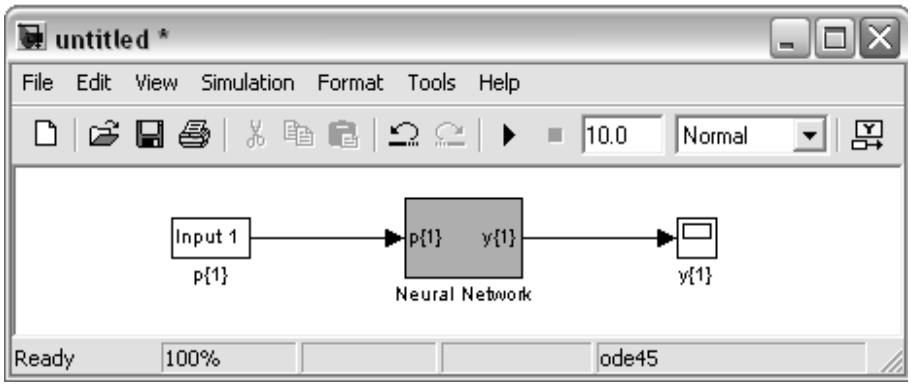


Рисунок 4.1 – Структурна схема мережі

Перегляньте всі елементи мережі, двічі клацнувши на них. Переконайтеся, що мережа складена правильно: має один вхід з вхідним вектором *p {1}* і один вихід з вихідним вектором *y {1}*, складається з двох шарів.

5. Перегляньте архітектуру шарів. Так прихований перший шар повинен мати такий вигляд (рис. 4.2):

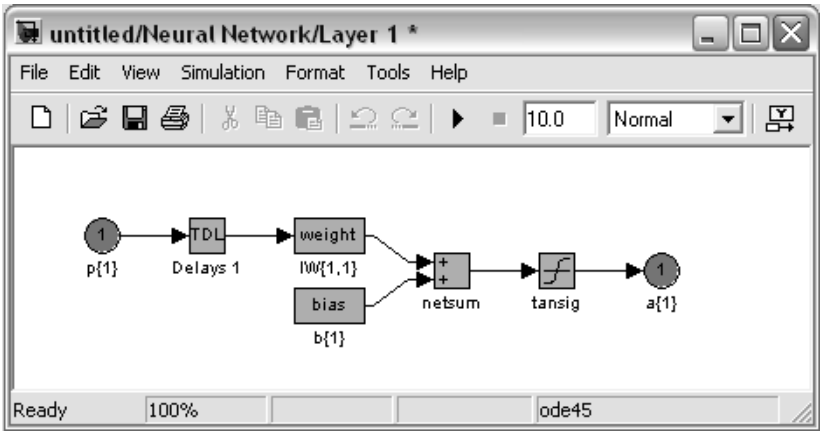


Рисунок 4.2 – Структурна схема мережі. Перший прихований шар мережі

На рис. 4.2 показано, що вхідний вектор *p{1}* подається на блок *TDL* – лінія затримки з ім'ям *Delays1*, яка забезпечує затримку вхідних сигналів на один такт. Лінію затримки додають до мережі для збільшення можливостей моделі. Сигнали після лінії затримки подаються на блок *weight* з ім'ям *IW{1, 1}* – масив комірок з матрицями ваг входів: матриці для шару 1 (прихованого) від вхідного вектора 1. До прихованого шару нейронів доданий блок зсуву з ім'ям *b {1}* – блок ваг зсувів, який призводить до прискорення процесу навчання. У модель включений блок *netsum* – блок підсумовування компонент для кожного нейрона першого шару: компонент від вектора входу вхідного шару з урахуванням затримки і зміщення. Вихідний сигнал сумматора надходить в нелінійний перетворювач, позначений спеціальним графічним символом, де перетворюється функцією активації *tansig,* і результат подається на вихід першого шару мережі *a{1}*.

6. Перегляньте структуру другого шару. Вихідний сигнал суматора другого шару перетворюється лінійною функцією активації *purelin*, і результат подається на вихід другого шару *a{2}*. Значення виходу другого шару подається на вихід мережі *y1*.

7. Необхідно навчити нейронну мережу за відсутності шуму. Параметри необхідно поставити такі:

net.trainParam.epochs=500; – задану кількість циклів навчання;

net.trainParam.show=50; – кількість циклів для показу проміжних результатів;

net.trainParam.goal=1e-5; – цільова помилка навчання;

p = [-1:0.05:1]; – вхідний вектор мережі;

*t = sin(2*pi*p);* – вихідний вектор мережі;

[net, tr] = train(net, p, t); – навчання мережі за відсутності шуму.

Процес навчання ілюструється графіком залежності помилки навчання від кількості циклів навчання (рис. 4.3).

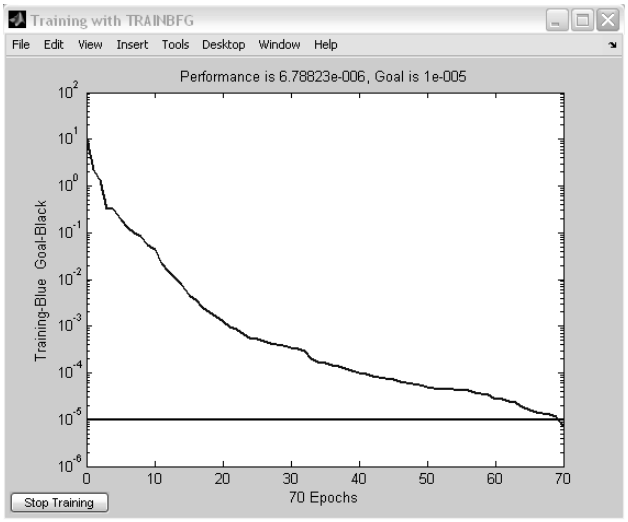


Рисунок 4.3 – Крива навчання нейронної мережі за відсутності шуму

8. Для створення нейронної мережі, не чутливої до впливу шуму, навчіть її із застосуванням одного ідеального і одного зашумленого векторів входу. Викривлений вектор має шум із середнім значенням 0,02, розподіленим за нормальним законом. Цільовий вектор складається з двох векторів. Такий спосіб навчання нейронної мережі дозволить правильно апроксимувати як зашумленную функцію, так і ідеальну:

$p_1 = [\,p \quad p + randn(size(p)) \cdot 0{,}02\,]$ – вхідний вектор, що складається з ідеального та зашумленого векторів;

$t_1 = [\,sin(2\pi \cdot p) \quad sin(2\pi \cdot p)\,]$ – цільової вектор, що складається з двох векторів.

9. Для реалізації нейронної мережі задайте в m-файлі наступні функції:

netn = net; % ініціалізація мережі;

netn.trainParam.epochs = 500; % задану кількість циклів навчання;

netn.trainParam.show = 50; % кількість циклів для показу проміжних результатів;

netn.trainParam.goal = 1e-3; % цільова помилка навчання;

*t1 = [sin(2*pi*p) sin(2*pi*p)];* % вихідний вектор мережі;

*p1 = [p, (p + randn (size(p))*0.02)];* % вхідний вектор мережі з шумом;

[netn, tr] = train (netn, p1, t1); % навчання мережі за наявності шуму.

Процес навчання зашумленої нейронної мережі ілюструється графіком залежності помилки навчання від кількості циклів навчання. Збережіть отриману криву навчання.

10. Для того щоб гарантувати правильність класифікації ідеальних векторів, нейронну мережу, навчену за наявності викривленого вектора, навчіть без шуму, задавши наступні параметри:

netn.trainParam.goal = 1e-5; % цільова помилка навчання;

[netn, tr] = train (netn, p, t); % повторне навчання мережі за відсутності шуму.

Проаналізуйте отриману криву повторного навчання нейронної мережі за відсутності шуму.

11. Після навчання нейронних мереж, перевірте ефективність їх функціонування.

12. Перевірка функціонування системи здійснюється наступним чином. Шум з діапазоном значень від 0 до 0,1 з кроком 0,01, розподіленим за нормальним законом, додається до вектора входу кожної мережі. Для кожного рівня шуму формується 20 зашумлених послідовностей (розмірність вхідного вектора дорівнює 1×1, а test = 20), потім обчислюється вихід мережі. Після цього визначається кількість помилкових класифікацій і обчислюється відсоток помилки. Наступний набір команд ілюструє описану процедуру:

tic % встановлює таймер

noise=0:0.01:0.1; % різні рівні шуму

```
test=20; % кількість ітерацій
network1=[];
network2=[];
t=sin(2*pi*p); % вихідний вектор
 for noiselevel = noise % організуємо цикл
    errors1=0; % початкове значення сумарної помилки мережі net
    errors2=0; % початкове значення сумарної помилки мережі  netn
     for i=1:test
       P = p + randn (size (p)) * noiselevel; % вхідний вектор
       A = sim (net, P); % використання мережі net
       errors1 = errors1+sum(sum(abs(A-t)))/2; % сумарна помилка мережі net
       An=sim(netn,P); % використання мережі netn
       errors2 = errors2 + sum(sum(abs(An-t)))/2; % сумарна помилка мережі netn
          echo off % вмикає повторення команд
      end
      network1 = [network1 errors1/41/20];
      network2 = [network2 errors2/41/20];
  end
  network1
  network2
toc
figure(1)
plot (noise,network1*100,'--b', noise, network2*100, ':r', 'LineWidth', 1.5);
% графіки залежностей відсотка помилки від рівня шуму
legend('відсоток помилки мережі net', 'відсоток помилки мережі netn');
 % легенда
xlabel('Рівень шуму','FontSize',12);
ylabel('Відсоток помилки','FontSize',12);
```

title ('Залежність відсотка помилки від рівня вхідного шуму', 'FontSize', 12, 'FontWeight', 'bold');

grid on

Для того щоб визначити ефективність функціонування кожної мережі, обчисліть відсоток помилки і побудуйте відповідний графік залежності відсотка помилки від рівня вхідного шуму, який зображений на рис. 4.4. На графіку видно, що чим більше рівень вхідного шуму, тим вище відсоток помилки.

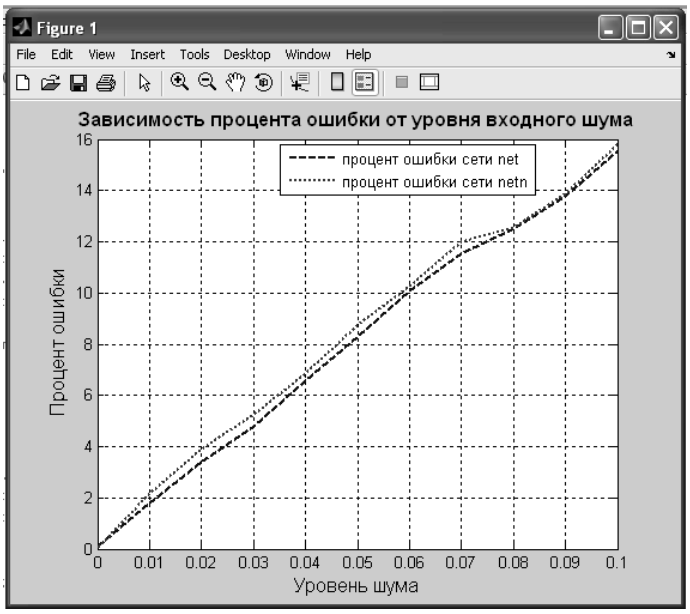


Рисунок 4.4 – Залежність відсотка помилки від рівня вхідного шуму

13. Проаналізуйте результати тестування, зробіть висновок про здатність різних нейронних мереж апроксимувати функції.

14. Застосуйте створену нейронну мережу для апроксимації даної функції. Для цього необхідно сформувати викривлений вектор входу із середнім значенням шуму рівними 0,03, розподіленими за нормальним законом. При використанні мережі задайте рівень зашумлення вхідного вектора більше, ніж при навчанні:

$$p_2 = p + randn(size(p)) \cdot 0{,}03.$$

15. Для того щоб застосувати навчену мережу для обробки даних, необхідно скористатися функцією *sim*.

an = sim (net, p); % використання нейронної мережі

16. Пропишіть наступний набір команд у файл програми:

*p2=randn(size(p))*0.03+p;* % вхідний вектор із шумом

[an, E]=sim(net, p2); % використання нейронної мережі

figure(2);

plot (p2, t, '+', p2, an, '-', p, t, ':', 'LineWidth', 1.5) % графіки

legend('вход','выход','sin(2\pi*t)');* % легенда

title ('Апроксимація функції y=sin(2\pi*t) двошаровою нейронною мережею', 'FontSize', 11.5, 'FontWeight', 'bold');* % заголовок

grid on % координатна сітка

E = mse (an-t) % середньоквадратична помилка використання нейронної мережі

17. У результаті ви маєте отримати графік апроксимації функції, що наведений на рис. 4.5.

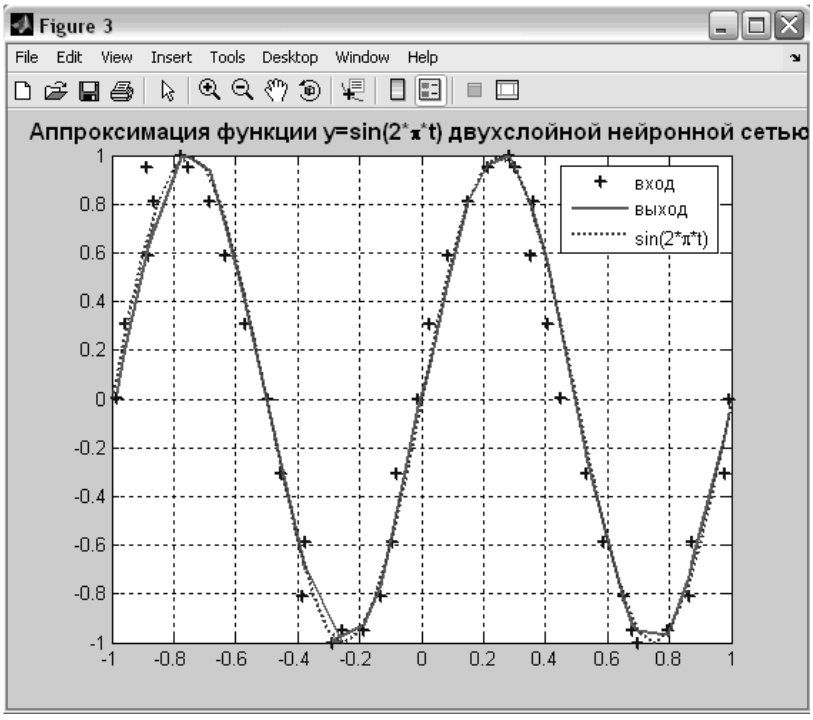


Рисунок 4.5 – Апроксимація функції $y = sin(2\pi x)$

двошаровою нейронною мережею

18. Проаналізуйте отримані графіки та зробіть висновки щодо роботи навченої нейронної мережі.

Зміст звіту

1. Тема й мета роботи.

2. Навести структурну схему розробленої мережі.

3. Навести отримані криві при навчанні нейронної мережі без шуму та за наявності шуму.

4. Зобразити графік залежності відсотка помилки від рівня вхідного шуму.

5. Висновки про здатність різних нейронних мереж апроксимувати функції.

Контрольні питання

1. Від чого залежить архітектура нейронної мережі?

2. Які етапи виділяють при моделюванні нейронної мережі?

3. Як задається вихідний вектор для мережі?

4. Які види передавальних функцій можуть обчислювати вихідний сигнал нейрона?

5. Поясніть алгоритм зворотного поширення помилки.

6. Назвіть критерії зупинки навчання нейронної мережі.

Література: [2, 6–9, 11–12].

Лабораторна робота № 5

Тема. Ознайомлення з принципами роботи нейронних мереж у складі систем керування

Мета: дослідити систему керування з нейромережевим регулятором на основі еталонної моделі.

Короткі теоретичні відомості

Пакет Neural Network Toolbox (нейронні мережі) містить засоби для проектування, моделювання, навчання та використання безлічі відомих парадигм апарата штучних нейронних мереж. Пакет може використовуватися при розв'язанні різноманітних задач, таких як обробка сигналів, нелінійне керування, фінансове моделювання та ін.

У пакеті Neural Network Toolbox реалізовані 3 архітектури нейронних мереж у вигляді наступних контролерів:

- контролер із прогнозом (NN Predictive Contrloller);
- контролер на основі моделі авторегресії з ковзним середнім (NARMA-L2 Controller);
- контролер на основі еталонної моделі (Model Reference Controller).

Застосування нейронних мереж для розв'язання завдань керування дозволяє виділити два етапи проектування:

- етап ідентифікації;
- етап синтезу закону керування.

На етапі ідентифікації розробляється модель керованого процесу у вигляді нейронної мережі, що на етапі синтезу використовується для синтезу регулятора. Для кожної з трьох архітектур використовується одна й та сама процедура ідентифікації, однак етапи синтезу істотно різняться.

При керуванні з прогнозом модель керованого процесу використовується для того, щоб пророчити його майбутню поведінку, а алгоритм оптимізації застосовується для розрахунку такого керування, що мінімізує різницю між бажаними й дійсними змінами виходу моделі.

При керуванні на основі моделі авторегресії з ковзаним середнім регулятор являє собою досить просту реконструкцію моделі керованого процесу.

При керуванні на основі еталонної моделі регулятор – це нейронна мережа, що навчена керувати процесом так, щоб він відслідковував поведінку еталонного процесу. При цьому модель керованого процесу активно використовується при настроюванні параметрів самого регулятора.

Динамічні моделі систем керування із нейромережевими регуляторами розміщені в спеціальному розділі Control System набору блоків NN Blocksets і включають три згадані вище моделі регуляторів, а також блок побудови графіків.

Оскільки жоден конкретний регулятор не є універсальним, то описані можливості всіх трьох типів регуляторів, кожний з яких має свої переваги й недоліки.

Регулятор з прогнозом. Цей регулятор використовує модель керованого процесу у вигляді нейронної мережі, для того щоб прогнозувати майбутні реакції процесу на випадкові сигнали керування. Алгоритм оптимізації обчислює керуючі сигнали, які мінімізують різницю між бажаними й дійсними змінами сигналу на виході моделі й у такий спосіб оптимізують керований процес. Побудова моделі керованого процесу виконується автономно з використанням нейронної мережі, що навчається в груповому режимі з використанням одного з алгоритмів навчання. Контролер, що реалізує такий регулятор, вимагає значного обсягу обчислень, оскільки для розрахунку оптимального закону керування оптимізація виконується на кожному такті керування.

Регулятор NARMA-L2. Із всіх архітектур цей регулятор вимагає найменшого обсягу обчислень. Даний регулятор – це просто деяка реконструкція нейромережевої моделі керованого процесу, отриманої на етапі ідентифікації. Обчислення в реальному часі пов’язані тільки з реалізацією нейронної мережі. Недолік методу полягає в тому, що модель процесу повинна бути задана в канонічній формі простору стану, якій відповідає супровідна матриця, що може приводити до обчислювальних похибок.

Регулятор на основі еталонної моделі. Необхідний обсяг обчислень для

цього регулятора можна зрівняти з попереднім. Однак архітектура регулятора з еталонною моделлю вимагає навчання нейронної мережі керованого процесу й нейронної мережі регулятора. При цьому навчання регулятора виявляється досить складним, оскільки навчання основане на динамічному варіанті методу зворотного поширення помилки. Перевагою регуляторів на основі еталонної моделі те що вони можуть застосовуватися до різних класів керованих процесів.

Порядок виконання роботи

1. Необхідно реалізувати систему керування, яка б керувала рухом ланки промислового робота, відслідковуючи вихід еталонної моделі:

$$\frac{d^2 y_r}{dt^2} = -9y_r - 6\frac{dy_r}{dt} + 9r, \qquad (5.1)$$

де y_r – вихід еталонної моделі; r – задає сигнал на вході моделі.

Структурна схема, що пояснює принцип побудови системи керування з еталонною моделлю, показана на рисунку 5.1.

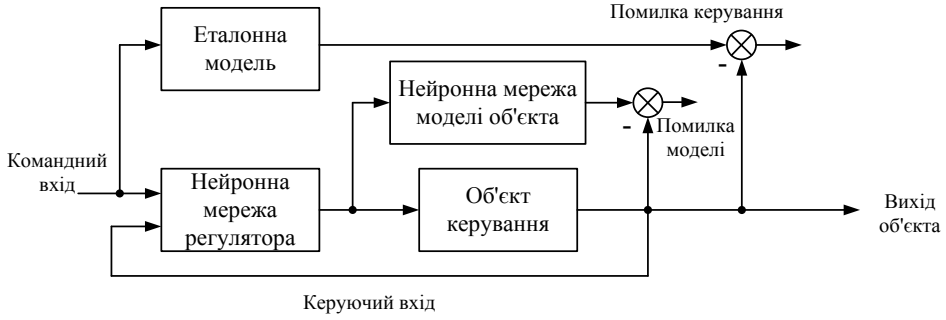


Рисунок 5.1 – Структурна схема

2. Побудуйте відповідну динамічну модель, реалізувавши її в Simulink.

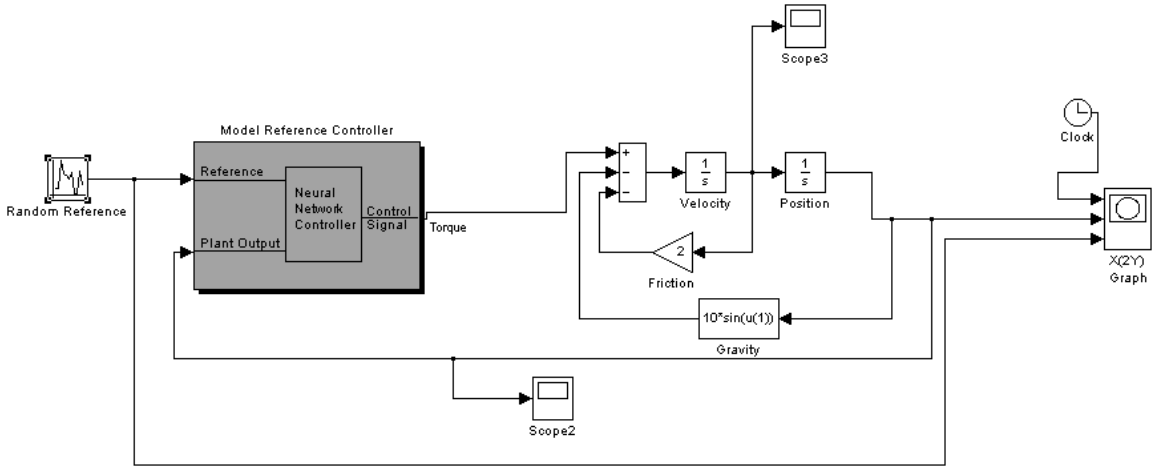


Рисунок 5.2 – Модель системи

3. Активуйте блок нейромережевого регулятора, двічі клацнувши на блок *Model Reference Controller*. Відкриється вікно графічного інтерфейсу контролера:

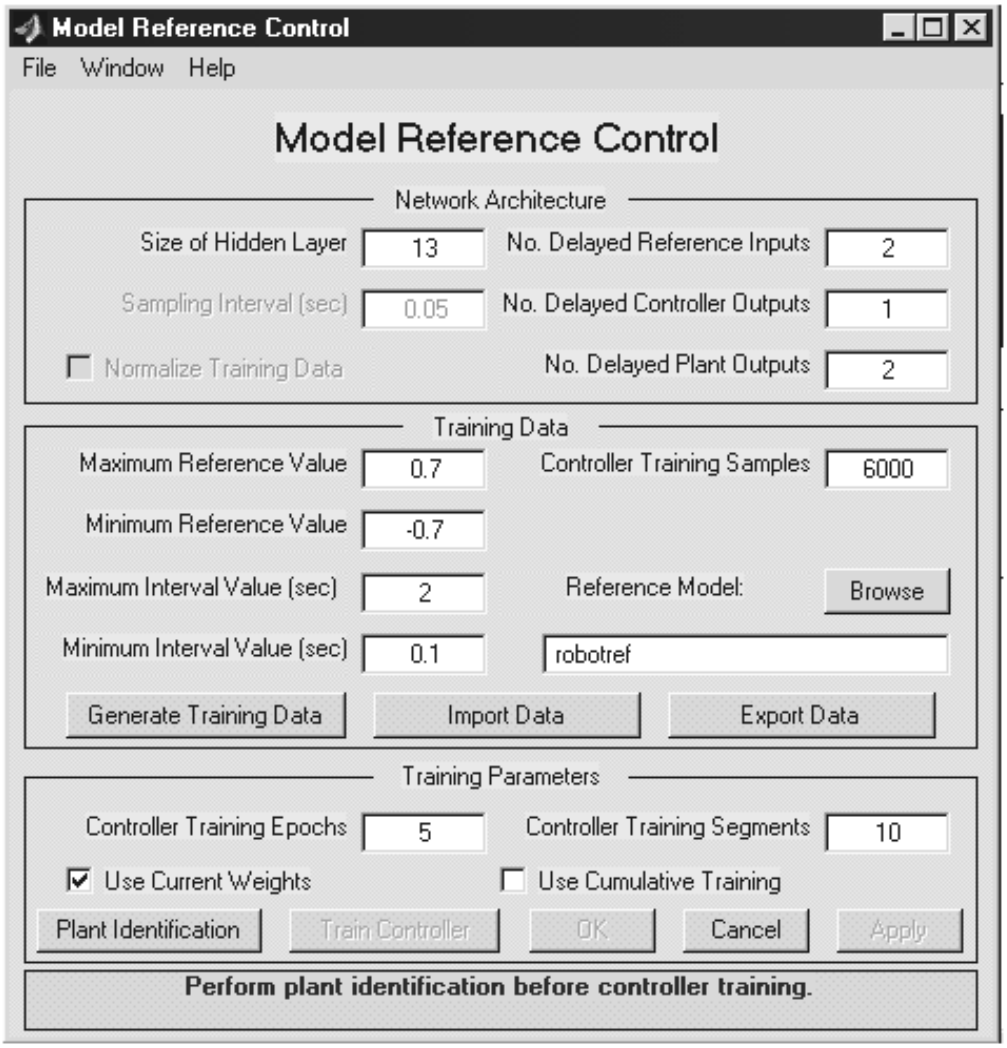


Рисунок 5.3 – Вікно блока нейромережевого регулятора

4. Перед тим як встановити параметри контролера, необхідно побудувати модель керованого процесу. Тобто необхідно виконати ідентифікацію керованого процесу – побудувати нейромережеву модель. Для цього відкрийте вікно спеціальної процедури *Plant Identification*, що зображене на рис. 5.4.

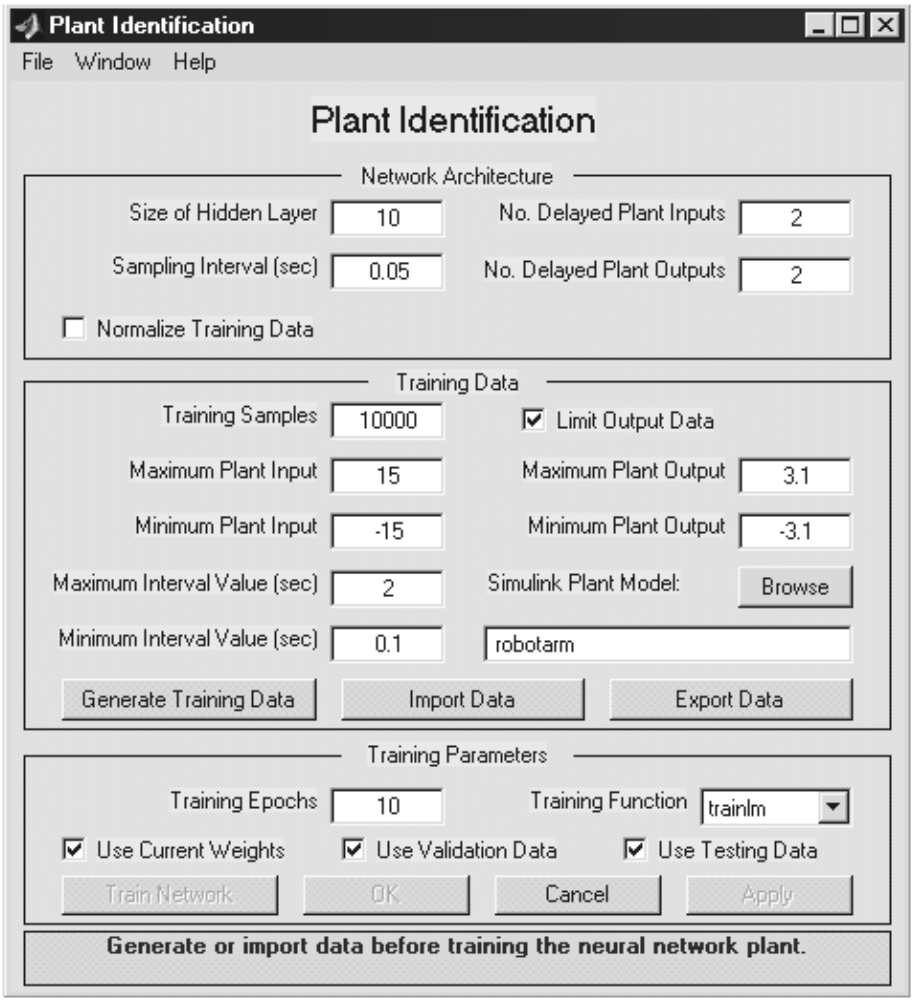


Рисунок 5.4 – Вид вікна *Plant Identification*

Це вікно може бути використане для побудови нейромережевих моделей для будь-якого динамічного об’єкта, що описаний моделлю Simulink.

5. Для системи керування необхідна динамічна модель, реалізована в Simulink, що відповідає рівнянню руху ланки, має вигляд, зображений на рис. 5.5. Реалізуйте наведену схему та збережіть її в робочому каталозі з ім’ям “*robotarm*”.

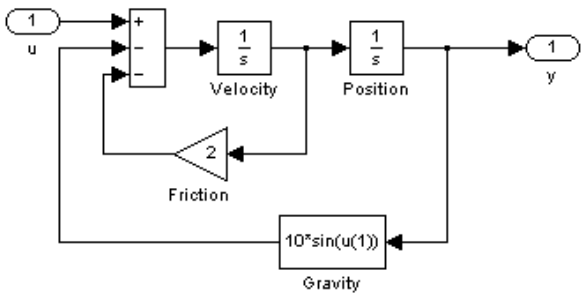


Рисунок 5.5 – Схема динамічної моделі ланки

6. Процедура ідентифікації потребує встановлення наступних параметрів:

– *Size of Hidden Layer* – розмір прихованого шару визначається кількістю використовуваних нейронів;

– *Sampling Interval* – такт дискретності в секундах визначає інтервал між двома послідовними моментами знімання даних;

– *No. Delayed Plant Inputs* – кількість елементів запізнення на вході моделі;

– *No. Delayed Plant Outputs* кількість елементів запізнення на виході моделі;

– *Normalize Trainig Data* – вікно контролю нормування навчальних даних до діапазону [0 1] ;

– *Trainig Samples* – довжина навчальної вибірки (кількість точок знімання інформації);

– *Maximum Plant Input* – максимальне значення вхідного сигналу;

– *Minimum Plant Input* – мінімальне значення вхідного сигналу;

– *Maximum Interval Value* – максимальний інтервал ідентифікації в секундах;

– *Minimum Interval Value* – мінімальний інтервал ідентифікації в секундах;

– *Limit Output Data* – вікно контролю, що дозволяє обмежити обсяг вихідних даних; тільки при включеному вікні контролю будуть доступні два наступних вікна редагування тексту;

– *Maximum Plant Output* – максимальне значення вихідного сигналу;

– *Minimum Plant Output* – мінімальне значення вихідного сигналу;

– *Simulink Plant Model* – завдання моделі Simulink із зазначенням вхідних і вихідних портів, що використовуються для побудови нейромережевої моделі керованого процесу;

– *Generate Trainig Data* – кнопка запуску процесу генерації навчальної послідовності;

– *Import Data* – імпорт навчальної послідовності з робочої області або файла даних;

– *Export Data* – експорт згенерованих даних у робочу область або файл;

– *Trainig Epochs* – кількість циклів навчання;

– *Trainig Function* – завдання навчальної функції;

– *Use Current Weights* – вікно контролю, що дозволяє підтвердити використання поточних ваг нейронної мережі;

– *Use Validation/Testing For Training* – вибір цих вікон контролю буде означати, що 25 відсотків даних з навчальної послідовності буде використано для формування тестової і контрольної підмножин відповідно.

7. Установивши параметри, як зазначено на рис. 5.4, натисніть кнопку *Generate Trainig Data*, буде запущена програма генерації навчальної послідовності. Програма генерує навчальні дані шляхом впливу випадкових східчастих впливів на модель Simulink керованого процесу. Графіки вхідного і вихідного сигналів об’єкта мають бути такими, як зображено на рис. 5.6.

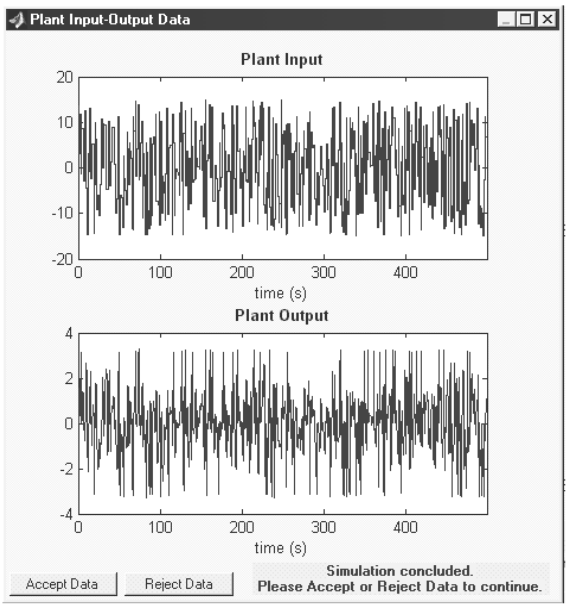


Рисунок 5.6 – Графіки вхідного і вихідного сигналів об’єкта

8. По завершенні генерації навчальної послідовності необхідно прийняти згенеровані дані (Accept Data) або відмовитися від них (Reject Data).

9. При поверненні до зміненого вікна *Plant Identification* починайте навчання нейронної мережі, натиснувши кнопку *Train Network*.

10. По завершенні навчання нейромережевої моделі на графіках відобразяться результати, а саме результати навчання та тестування на контрольній пі-

дмножині (*Training data, Testing data*). Для того щоб згенерувати або імпортувати нові дані, продовжити навчання або зберегти отримані результати, натисніть кнопку *Ok* або *Apply*. У результаті параметри нейромережевої моделі керованого процесу будуть занесені до блока *Model Reference Controller*.

11. Для навчання контролера в полі *Reference Model* необхідно вказати ім'я схеми, що описує еталонну модель. Для цього слід рівняння еталонної моделі (5.1) методом зниження похідної реалізувати в Simulink моделі та зберегти її в робочому каталозі з ім'ям "*robotref*".

12. Самостійно згенеруйте параметри контролера на навчіть мережу.

13. Навчіть регулятор на основі нейронної мережі. Установіть кількість циклів навчання за вказівками викладача. Проведіть 5000 вимірів навчальної послідовності та виведіть результати на екран.

14. Порівняйте роботу моделі з різними параметрами, отримавши графіки їх роботи.

Зміст звіту

1. Указати номер, тему й мету лабораторної роботи.

2. Навести відповідні вікна з параметрами моделі керованого процесу та контролера.

3. Відобразити отримані графіки вхідних і вихідних сигналів об'єкта.

4. Зробити порівняльні висновки стосовно роботи нейромережевого контролера при різних параметрах.

Контрольні питання

1. Які архітектури нейронних мереж реалізовані в пакеті Neural Network Toolbox? Охарактеризуйте кожну з них.

2. Як побудувати нейромережеву модель?

3. Що таке ідентифікація керованого процесу?

4. Які обов'язкові параметри вказують при створенні нейромережевої моделі?

5. Як навчається мережа?

Література: [2, 6–9, 11–12].

Лабораторна робота № 6

Тема. Генетичні алгоритми

Мета: навчитися реалізовувати генетичні алгоритми в додатку Genetic Algorithm and Direct Search Toolbox та застосовувати їх у завданнях оптимізації.

Короткі теоретичні відомості

Генетичний алгоритм (англ. genetic algorithm) – це евристичний алгоритм пошуку, що використовується для розвязання завдань оптимізації й моделювання шляхом випадкового підбору, комбінування й варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію, є різновидом еволюційних обчислень. Відмінною рисою генетичного алгоритму є акцент на використанні оператора «схрещування», що виконує операцію рекомбінації рішень-кандидатів, роль якої аналогічна ролі схрещування в живій природі.

Функція придатності – це функція, що підлягає оптимізації. У випадку стандартних оптимізаційних алгоритмів вона відома як цільова функція. Функцію придатності можна записати в М-файл і передати у вигляді якогось аргументу в основний генетичний алгоритм.

Індивідуум – деяка точка, у якій можливий розрахунок функції придатності. Значення функції придатності індивідуалізованого об'єкта саме і є її кількісний показник. Наприклад, якщо функція придатності має такий вигляд:

$$f(x1,x2,x3) = (2x1 + 1)^2 + (3x2 + 4)^2 + (x3 – 2)^2,$$

то вектор (2, 3, 1), розмірність якого дорівнює числу змінних даної задачі, і є індивідуум. Кількісний показник індивідуума як об'єкта (2, 3, 1) буде f(2, -3, 1) = 51. Об'єкт індивідуума іноді називається *геном*, а компоненти вектора називаються генами.

Сімейство – це масив індивідуалізованих об'єктів. Наприклад, якщо розмір сімейства дорівнює 100 і число змінних у функції придатності дорівнює 3, тоді дане сімейство являє собою матрицю розміром 100x3. Той самий об'єкт індивідуума може з'являтися в даному сімействі більше одного разу.

Для того щоб одержати нове покоління, в генетичному алгоритмі на ко-

жній ітерації проводиться деяка серія розрахунків на основі поточного сімейства. Кожне успішне сімейство називається новим поколінням. Для того щоб одержати наступне сімейство, у генетичному алгоритмі виконується відбір певних індивідуалізованих об'єктів у поточному сімействі, що називаються батьківськими і які далі використовуються для утворення об'єктів індивідуумів для наступного сімейства – дочірнього. Як правило, вибираються ті батьки, які мають більше значення функції придатності.

Genetic Algorithm and Direct Search Toolbox

Genetic Algorithm and Direct Search Toolbox призначений для розширення функціональних можливостей пакета MATLAB і, зокрема, Optimization Toolbox новими видами алгоритмів оптимізації. Подібні методи й алгоритми найчастіше використовуються у випадку, коли цільова функція є переривчастою, істотно нелінійною, стохастичною і не має похідних або ці похідні є недостатньо визначеними. Genetic Algorithm and Direct Search Toolbox допоможе розв’язати задачі, які або неможливо розв’язати звичайними методами, або виникають нестійкі рішення при застосуванні стандартних математичних методів.

Genetic Algorithm and Direct Search Toolbox містить у собі програми для розв’язання задач оптимізації на основі використання наступних методів:

- генетичний алгоритм;
- метод прямого пошуку.

Завданням даного тулбоксу є пошук мінімуму функції придатності.

Усі функції є М-файлами пакета MATLAB, складені з операторів MATLAB і являють собою реалізацію спеціалізованих алгоритмів оптимізації. Є можливість переглядати ці функції шляхом виконання звичайного оператора:

type function_name.

Генетичний алгоритм – це метод розв’язання задач оптимізації на основі природного добору, аналогічно тому, як це відбувається в процесі біологічної еволюції. У генетичному алгоритмі відбувається багаторазова модифікація сімейства окремих розв’язків. На кожному кроці проводиться відбір вибраних

навмання суб'єктів з отриманого поточного розв’язання, що називається батьківським і яке використовується для генерації наступного дочірнього покоління. За допомогою послідовного відбору поколінь відбувається "еволюція" у напрямку до оптимального розв’язання. Генетичний алгоритм можна застосовувати до різноманітних задач оптимізації, які недостатньо чітко вписуються в стандартні оптимізаційні алгоритми.

На кожному кроці для генерації наступного покоління в генетичному алгоритмі використовуються наступні три основні правила:

– *правило відбору* – відбирає об'єкти, що називаються батьківськими та які становлять основу наступного покоління з поточного розв’язання;

– *правило схрещування*, за яким відбувається вибір із двох батьківських об'єктів дочірнього для наступного покоління;

– *правило мутації*, за яким на основі ймовірнісних алгоритмів з батьківських об'єктів формуються дочірні.

Генетичний алгоритм відрізняється від стандартних методів оптимізації наступними особливостями (табл. 6.1):

Таблиця 6.1 – Порівняння стандартного й генетичного алгоритмів

Стандартний алгоритм	Генетичний алгоритм
На кожній ітерації генерується одна єдина точка. Отримана послідовність точок сходиться до оптимального рішення.	На кожній ітерації генерується сімейство точок. Отримане сімейство сходиться до оптимального розв’язання.
Вибір наступної точки здійснюється на основі детермінованих розрахунків.	Вибір наступного сімейства точок здійснюється на основі стохастичних розрахунків.

Структура алгоритму

Генетичний алгоритм працює відповідно до наступної схеми:

1) для організації початку рахунку створюється довільне вихідне сімейство;

2) алгоритм робить деяку послідовність нових сімейств або поколінь. На кожному окремому кроці алгоритм використовує певні індивідууми з поточно-

го покоління, для того щоб створити наступне покоління. При формуванні нового покоління в алгоритмі проводяться наступні дії:

– відмічається кожний член поточного сімейства за допомогою обчислення відповідного значення придатності;

– проводиться масштабування отриманого ряду значень функції придатності, що дозволяє побудувати діапазон значень більш зручний для наступного використання;

– вибираються батьківські значення на основі значень їхньої придатності;

– частина індивідуумів з батьківського покоління має більші значення функції придатності, які далі вибираються як елітні значення. Ці елітні значення передаються вже в наступне покоління;

– дочірні значення утворюються або шляхом якихось випадкових змін окремого одного батька – *мутація*, або шляхом комбінації векторних компонентів якоїсь пари батьків – *кросовер*;

– заміна поточного сімейства на дочірнє з метою формування наступного покоління;

3) зупинка алгоритму здійснюється тоді, коли виконується будь-який критерій зупинки.

Графічний інструментарій генетичного алгоритму

Інструментарій генетичного алгоритму – це фактично графічний інтерфейс користувача, за допомогою якого користувач має можливість роботи з генетичним алгоритмом без звернення до нього з командного рядка. Для звернення до *Інструментарію генетичного алгоритму* варто виконати команду *gatool*, при виконанні якої відкривається наступний інструментарій (рис. 6.1):

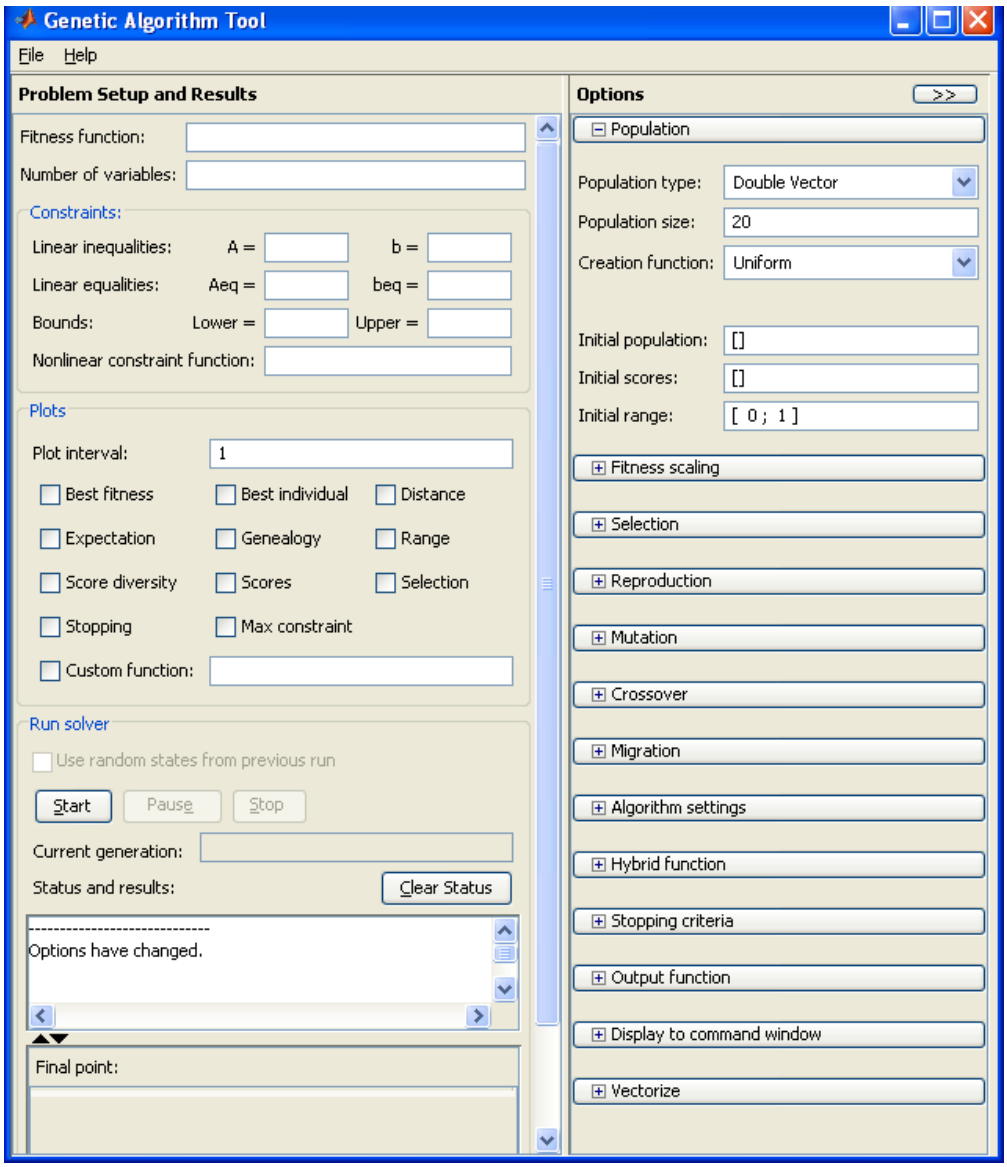


Рисунок 6.1 – Графічний інструментарій

Для роботи з *Інструментарієм генетичного алгоритму* варто ввести наступну інформацію:

– *Fitness function* – підлягаюча мінімізації цільова функція. У форму @fitnessfun вводиться функція придатності, де fitnessfun.m – це M-файл для розрахунку функції придатності. Знак «@» створює описувач функції для fitnessfun.

– *Number of variables* – розмір вхідного вектора для функції придатності.

Для виконання генетичного алгоритму варто клікнути мишкою на кнопку "*Start*". Далі в інструментарії в панелі "*Status and Results*" здійснюється відо-

браження результатів оптимізації.

За допомогою панелі "*Options*" можливо настроїти опції генетичного алгоритму. Для огляду опцій для заданого режиму в даній панелі варто клікнути на знак "+".

Основними параметрами в GATool є:

- популяція (вкладка "Population");
- масштабування (вкладка "Fitness Scaling");
- оператор відбору (вкладка "Selection");
- оператор репродукції (вкладка "Reproduction");
- оператор мутації (вкладка "Mutation");
- оператор схрещування (вкладка "Crossover");
- перенесення особин між популяціями (вкладка "Migration");
- спеціальні параметри алгоритму (вкладка "Algorithm settings");
- задання гібридної функції (вкладка "Hybrid function");
- задання критерію зупинки алгоритму (вкладка "Stopping criteria");
- відображення різної додаткової інформації у ході роботи генетичного алгоритму (вкладка "Plot Functions");
- відображення результатів роботи алгоритму у вигляді нової функції (вкладка "Output function");
- задання набору інформації для виведення в командне вікно (вкладка "*Display to command window*");
- спосіб обчислення значень оптимізованої й обмежувальної функцій (вкладка "*User function evaluation*").

У вкладці настроювання популяцій ("*Population*") користувач має можливість вибрати тип математичних об'єктів, до якого будуть належать особини всіх популяцій (подвійний вектор, бітовий рядок або користувацький тип). При цьому варто враховувати, що використання бітового рядка й користувацьких типів накладають обмеження на перелік припустимих операторів створення, мутації й схрещування особин. Також вкладка популяції дозволяє налаштову-

вати розмір популяції (*Population size* – зі скількох особин буде складатися кожне покоління) і яким чином буде створюватися початкове покоління (*Creation function: Uniform* – якщо відсутні обмеження, в іншому випадку – *Feasible population*). Крім того, є можливість задати вручну початкове покоління (використовуючи пункт *Initial population*) або його частину, початковий рейтинг особин (пункт *Initial scores*), а також задати обмежувальний числовий діапазон, якому повинні належати особини початкової популяції (*Initial range*).

У вкладці масштабування ("*Fitness Scaling*") зазначається функція масштабування, яка конвертує значення, що досягаються оптимізуючою функцією в значення, що лежать у межах, припустимих для оператора відбору. При виборі як функції масштабування параметра *Rank* масштабування буде приводитися до рейтингу, тобто особинам присвоюється рейтинговий номер (для кращої особини – одиниця, для наступної – двійка і так далі). Пропорційне масштабування (*Proportional*) задає ймовірності пропорційно заданому числовому ряду для особин. При виборі опції *Top* найбільше рейтингове значення присвоюється відразу декільком найбільш значущим особинам (їхнє число вказується у вигляді параметра). При виборі масштабування типу *Shift linear* є можливість указати максимальну ймовірність найкращої особини.

Вкладка "*Selection*" дозволяє вибрати оператор відбору батьківських особин на основі даних з функції масштабування. Як доступні для вибору варіантів оператора відбору пропонуються наступні:

– *Tournament* – випадково вибирається зазначене число особин, серед них на конкурсній основі вибираються кращі;

– *Roulette* – імітується рулетка, у якій розмір кожного сегмента встановлюється відповідно до його ймовірності;

– *Uniform* – батьки вибираються випадковим чином відповідно до заданого розподілу й з урахуванням кількості батьківських особин і їхніх імовірностей;

– *Stochastic uniform* – будується лінія, у якій кожному батькові ставиться у відповідність її частина певного розміру (залежно від імовірності батька),

потім алгоритм пробігає по лінії кроками однакової довжини й вибирає батьків залежно від того, на яку частину лінії потрапив крок.

Вкладка репродукції ("*Reproduction*") уточнює, яким чином відбувається створення нових особин. Пункт *Elite count* дозволяє вказати число особин, які гарантовано перейдуть до наступного покоління. Пункт *Crossover fraction* указує частину особин, які створюються шляхом схрещування. Інша частина створюється шляхом мутації.

У вкладці оператора мутації ("*Mutation*") вибирається тип оператора мутації. Доступні наступні варіанти:

– *Gaussian* – додає невелике випадкове число (відповідно до розподілу Гаусса) до всіх компонентів кожного вектора-особини;

– *Uniform* – вибираються випадковим чином компоненти векторів і замість них записуються випадкові числа із допустимого діапазону;

– *Adaptive feasible* – генерує набір напрямків залежно від останніх найбільш удалих і невдалих поколінь і з урахуванням обмежень, що накладаються, просувається вздовж усіх напрямків на різну довжину;

– *Custom* – дозволяє задати власну функцію.

Вкладка схрещування ("*Crossover*") дозволяє вибрати тип оператора схрещування: одноточкове, двоточкове, евристичне, арифметичне або розсіяне.

У вкладці "*Migration*" можна задавати правила, згідно з якими особини будуть переміщатися між підпопуляціями в межах однієї популяції. Підпопуляції створюються, якщо як розмір популяції зазначений вектор, а не натуральне значення. У даній вкладці можна вказати напрямок міграції (*forward* – у наступну підпопуляцію, *both* – у попередню й наступну), частину мігруючих особин (*Fraction*) і частоту міграції (скільки поколінь проходить між міграціями).

Вкладка спеціальних опцій алгоритму ("*Algorithm settings*") дозволяє задавати параметри розв’язання системи нелінійних обмежень, що накладаються на алгоритм.

Вкладка "*Hybrid function*" дозволяє задати ще одну функцію мінімізації, що буде використовуватися після закінчення роботи алгоритму. Як можливі гі-

бридні функції доступні наступні вбудовані в саме середовище MATLAB функції:

– *none* (не використовувати гібридну функцію);

– *fminsearch* (пошук мінімального зі значень);

– *patternsearch* (пошук за зразком);

– *fminunc* (для необмеженого алгоритму);

– *fmincon* (для алгоритму із заданими обмеженнями).

У вкладці критерію зупинки ("*Stopping criteria*") зазначаються ситуації, при яких алгоритм робить зупинку. При цьому задаються наступні параметри:

– *Generations* – максимальне число поколінь, після перевищення якого відбудеться зупинка;

– *Time limit* – ліміт часу на роботу алгоритму;

– *Fitness limit* – якщо оптимізоване значення менше або дорівнює даному ліміту, то алгоритм зупиниться;

– *Stall generations* – алгоритм зупиняється у випадку, якщо немає поліпшення для цільової функції в послідовності наступних один за одним поколінь довжиною Stall generations;

– *Stall time limit* – алгоритм зупиняється у випадку, якщо немає поліпшення для цільової функції протягом інтервалу часу в секундах рівного Stall time limit;

– *Function tolerance* і *Nonlinear constraint tolerance* – мінімальні значення змін оптимізованої та обмежувальної функцій відповідно, при яких алгоритм продовжить роботу.

Вкладка "*Plot Functions*" дозволяє вибирати різну інформацію, що виводиться у ході роботи алгоритму й показує як коректність його роботи, так і конкретні результати, що досягаються алгоритмом. Найбільш важливими й використовуваними для відображення параметрами є:

– *Plot interval* – число поколінь, по закінченні якого відбувається чергове відновлення графіків;

– *Best fitness* – відображення найкращого значення оптимізованої функції для кожного покоління;

– *Best individual* – відображення найкращого представника покоління при найкращому результаті в кожному з поколінь;

– *Distance* – відображення інтервалу між значеннями особин у поколінні;

– *Expectation* – виводить ряд імовірностей і відповідні їм особини поколінь;

– *Genealogy* – відображення генеалогічного дерева особин;

– *Range* – відображення найменшого, найбільшого й середнього значень оптимізованої функції для кожного покоління;

– *Score diversity* – вивести гістограму рейтингу в кожному поколінні;

– *Scores* – відображення рейтингу кожної особини в поколінні;

– *Selection* – відображення гістограми батьків;

– *Stopping* – відображення інформації про стан усіх параметрів, що впливають на критерії зупинки;

– *Custom* – відображення на графіку деякої зазначеної користувачем функції.

Вкладка виведення результатів у вигляді нової функції ("*Output function*") дозволяє включити виведення історії роботи алгоритму в окремому вікні із заданим інтервалом поколінь (прапор *History to new window* і поле *Interval* відповідно), а також дозволяє задати й вивести довільну вихідну функцію, що задається в поле *Custom function*.

Вкладка "*User function evaluation*" описує, у якому порядку відбувається обчислення значень оптимізованої та обмежувальної функцій (окремо, паралельно в одному виклику або одночасно).

Вкладка "*Display to command window*" дозволяє задавати інформацію, що відображається в основному командному вікні MATLAB при роботі алгоритму. Можливі наступні значення: *Off* – немає виведення в командне вікно, *Iterative* – виведення інформації про кожну ітерацію працюючого алгоритму, *Diagnose* –

виведення інформації про кожну ітерацію й додаткові відомості про можливі помилки й змінені ключові параметри алгоритму, *Final* – виводиться тільки причина зупинки й кінцеве значення.

Порядок виконання роботи

1. Необхідно знайти мінімум наступної функції:

$$y(x_1, x_2) = x_1^2 - 2x_1x_2 + 6x_1 + x_2^2 - 6x_2 .$$

2. Для створення m-файла, що необхідний для розрахунку даної функції, слід створити порожній m-файл і ввести наступний код:

```
function z = my_fun(x)
z = x(1)^2 - 2*x(1)*x(2) + 6*x(1) + x(2)^2 - 6*x(2);
```

3. Зберегти m-файл у поточній робочій директорії MATLAB з ім'ям my_fun. m.

4. Для перевірки, що М-файл повертає точне рішення, варто виконати:

```
my_fun([2 3])
ans =
     -5
```

5. Відкрити інструментарій генетичного алгоритму. У поле *Fitness function* ввести ім'я цільової функції @my_fun, указати розмірність вхідного вектора для функції придатності.

6. Установити значення параметрів генетичного алгоритму: кількість особин у популяції = 10, кількість поколінь = 100 (у вкладці критерію зупинки алгоритму), початковий відрізок = [-1; 1].

7. У розділі *Plots* установити прапорці для графіків *Best fitness, Best individual, Distance.* Запустити генетичний алгоритм, проаналізувати отримані графіки.

8. У результаті завершення процесу у вікні *Final point* з'явиться значення змінної *x*, що відповідає мінімуму функції, а у вікні *Status and result* можна побачити знайдене мінімальне значення цільової функції.

9. Знайти максимум функції:

$$y(x) = 3\sqrt[3]{(x+4)^2} - 2x - 8 .$$

10. Відобразити отримані графіки.

11. Перевірити правильність розв’язання алгоритму, побудувавши графік заданої функції та порівняти точки мінімуму (максимуму).

Зміст звіту

1. Тема й мета роботи.

2. Навести отримані результати для функцій: результати роботи генетичного алгоритму, графіки функцій.

3. Зробити порівняльні висновки стосовно роботи генетичного алгоритму з різними функціями.

4. Висновки з роботи.

Контрольні питання

1. Дайте визначення *генетичного алгоритму*.

2. Що таке функція пристосовуваності?

3. Назвіть критерії зупинки роботи генетичного алгоритму.

4. Яким чином створюється початкова популяція?

5. Чому значення найкращого представника покоління відрізняється від попереднього?

Література: [9, 11–12].

КРИТЕРІЇ ОЦІНЮВАННЯ

При оцінюванні роботи студентів на лабораторних заняттях, враховується їх присутність та активність при виконанні завдання. Під час захисту виконаної лабораторної роботи враховується якість оформлення звіту (наявність теми, мети, висновків, графіків, пояснень, тощо), своєчасність здачі та володіння матеріалом.

Для студентів повної форми навчання навчальним планом передбачено 8 занять та 6 лабораторних робіт різної складності, їх знання оцінюються згідно з таблицею:

№	Присутність на занятті (консультації)	Активність на занятті	Якість оформлення звіту	Своєчасність здачі	Якість захисту	Кількість балів
1.	0,5	2,0	0,5	0,2	2,0	5,2
2.	0,5	2,0	0,5	0,2	2,3	5,5
3.	0,5	2,0	0,5	0,2	2,7	5,9
4.	0,5	2,0	0,5	0,2	2,8	6,0
5.	0,5	2,0	0,5	0,2	2,8	6,0
6.	0,5	2,0	0,5	0,2	2,8	6,0
7.	0,5	2,0				2,5
8.	0,5	2,0				2,5
Бали за всі заняття		22,5	Сумарний бал за всі лабораторні роботи			39,6

Для студентів скороченої форми навчання навчальним планом передбачено 9 занять та 6 лабораторних робіт різної складності, їх знання оцінюються згідно з таблицею:

№	Присутність на занятті (консультації)	Активність на занятті	Якість оформлення звіту	Своєчасність здачі	Якість захисту	Кількість балів
1.	0,5	2,0	0,5	0,2	3,0	6,2
2.	0,5	2,0	0,5	0,2	3,2	6,4
3.	0,5	2,0	0,5	0,2	3,5	6,7
4.	0,5	2,0	0,5	0,2	3,8	7,0
5.	0,5	2,0	0,5	0,2	3,8	7,0
6.	0,5	2,0	0,5	0,2	4,0	7,2
7.	0,5	2,0				
8.	0,5	2,0				
9.	0,5	2,0				
Бали за всі заняття		22,5	Сумарний бал за всі лабораторні роботи			48,00

СПИСОК ЛІТЕРАТУРИ

1. Андрейчиков А. В. Интеллектуальные информационные системы / А. В. Андрейчиков, О. Н. Андрейчикова. – Москва : “Финансы и статистика”, 2004. – 424 с.

2. Брюхомицкий Ю. А. Нейросетевые модели для систем информационной безопасности / Ю. А. Брюхомицкий. – Таганрог : Изд-во ТРТУ, 2005. – 160 с.

3. Тим Джонс М. Программирование искусственного интеллекта в приложениях / М. Тим Джонс ; пер. с англ. А. И. Осипов – Москва : ДМК Пресс, 2006. – 312 с.

4. Люгер Джордж Ф. Искусственный интеллект: стратегии и методы решения сложных проблем / Ф. Люгер Джордж. – 4-е изд. – Москва : “Вильямс”, 2003. – 864 с.

5. Ясницкий Л. Н. Введение в искусственный интеллект / Л. Н. Ясницкий. – Москва : «Академия», 2008. – 176 с.

6. Поспелов Д. А. Нечеткие множества в моделях управления и искусственного интеллекта / Д. А. Поспелов. – Москва : Наука, 1986. – 312 с.

7. Яхъяева Г. Э. Нечеткие множества и нейронные сети / Г. Э. Яхъяева. – Москва : Бином, 2006. – 316 с.

8. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. – Москва : Телеком, 2006. – 452 с.

9. Дьяконов В. Математические пакеты расширения MATLAB : специальный справочник / В. Дьяконов, В. Круглов. – СПб. : Питер, 2001. – 480 с.

10. Джексон Питер. Введение в экспертные системы / Питер Джексон. – 3-е изд. – 2001. – 624 с.

11. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / Г. К. Вороновский, К. В. Махотило, С. Н. Петрашев, С. А. Сергеев. – Харьков : ОСНОВА, 1997. – 112 с.

12. Дьяконов В. П. MATLAB 6.5 SP1/7/7 SP2+ Simulink 5/6. Инструменты искусственного интеллекта и биоинформатики / В. П. Дьяконов, В. В. Круглов. – Москва : СОЛОН-ПРЕСС, 2006. – 456 с.