МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Звенігородський О.С., Зінченко О.В., Чичкарьов Є.А., Березівський М.Ю.

Штучний інтелект

Вступний курс

Методичний посібник до практичних занять для студентів спеціальнстей: 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки», 123 «Комп'ютерна інженерія», 124 «Системний аналіз», 125 «Кібербезпека та захист інформації», 126 «Інформаційні системи та технології»

Київ 2023

УДК 681.324

Рекомендовано на засіданні вченої ради Навчально-наукового інституту інформаційних технологій (Протокол № 2 від 13.09.2023 року)

Звенігородський О.С., Зінченко О.В., Чичкарьов Є.А., Березівський М.Ю. Штучний інтелект. Вступний курс. Методичний посібник. – К.: ДУІКТ, 2023. – 74 с.

Рецензенти: **Савченко В.А**., доктор технічних наук, професор, директор Навчально-наукового інституту Захисту інформації Державного університету інформаційно-комунікаційних технологій.

Корнага Я.І., доктор технічних наук, доцент, доцент кафедри технічної безпеки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Методичний посібник до практичних занять призначений для отримання студентами програмних результатів навчання з дисципліни «Штучний інтелект». Предметом дослідження є моделі, алгоритми і методи штучного інтелекту, що застосовуються для вирішення практичних задач. Студенти мають можливість дослідити продукційні молелі, системи нечіткоко виведення, нейронні мережі і генетичний алгоритмом. У методичному посібнику в кожному завданні наведено достатньо практичних прикладів для полегшення вивчення матеріалу.

УДК 681.324

Зміст

Практична робота 1
Створення бази фактів і бази знань засобами SWI-Prolog.

Мета. Вивчити принципи продукційних систем логічного виведення, засвоїти основи мови програмування Prolog і порядок роботи в online-середовищі SWI-Prolog.

Завдання

1. Згідно з варіантом створити базу фактів і декілька правил бази знань для предметної області (табл. 1).
2. Зв'язки предметної області подати у виді блок-схеми.
3. Створити SWI-Prolog програму.
4. Навести приклади п'яти типів запитань до створеної бази фактів і знань.
5. Проаналізувати отримані результати і зробити висновки.

Примітка. Номер варіанта співпадає з номером зі списку групи. Якщо номер зі списку перевищує кількість варіантів, варіант визначається як ціле число залишку від ділення номера зі списку на кількість варіантів. Якщо номер зі списку кратний кількості варіантів, то обирається останній варіант. Приклад: номер у списку – 31, варіантів 22 то варіант – 9 (31/22 =1 залишок 9).
номер у списку – 22, варіантів 22 то варіант – 1

Таблиця 1.1 – Перелік варіантів предметної області

№ з/п	Предметна область
1.	Клуб за інтересами (прихильники письменників)
2.	Клуб за інтересами (любителі собак)
3.	Друзі і вороги
4.	Адміністративна структура компанії
5.	Генеалогічне дерево київських князів
6.	Структура спортивної команди
7.	Простір доменних імен DNS (Domain Name System)
8.	Адміністративна структура магазину.
9.	Класифікація музичних інструментів
10.	Класифікація спортивних ігор.
11.	Класифікація медичних препаратів
12.	Звання в армії.
13.	Тварини: хижаки і жертви
14.	Країни: столиця, населення
15.	Адміністративна структура ДУТ
16.	Навчальний процес: викладач, асистент, лекція, практика.
17.	Навчальний процес: дисципліна, викладач, група, студент
18.	Склад медичного персоналу лікарні
19.	Склад спортивної команди Динамо Київ.
20.	Компанії та продукти які вони випускають
21.	Адміністративна структура школи
22.	Адміністративна структура України

Зміст звіту

1. Титульна сторінка.
2. Тема практичного заняття.
3. Мета заняття.
4. Завдання.
5. Опис виконання завдань по пунктам з наданням блок-схем і скріншотів.
6. Текст SWI-Prolog програми
7. Висновки.

Контрольні питання.

1. Продукційна модель подання знань
2. Типи речень в Prolog.
3. Механізм логічного виводу в Prolog.
4. Типи запитань в Prolog.

Теоретичні відомості.

1.1. Основні поняття Prolog

Prolog – це мова програмування для символічних, нечислових обчислень. Вона особливо добре пристосована для вирішення проблем, що стосуються об’єктів та відносин між об’єктами. Програма на Prolog складається з речень (тверджень). Кожне речення закінчується крапкою. В загальному випадку речення складається із імені і тіла (предиката). Предикат записується як деякий вираз в круглих дужках перед яким без пропуска записується ім’я предиката. Наприклад, відносини між об’єктами студент з прізвищем Marchenko і групою KND31, де він навчається, мовою Prolog можна записати наступним чином:

student(marchenko, knd31).

Всі складові Prolog є предикатами, в тому числі функції і оператори (арифметичні, оператори вводу/виводу, розгалуження і т.і.). Це не зовсім зручно для програмування алгоритмічних дій, але головною перевагою Prolog перед іншими мовами є вбудований алгоритм логічного виведення, що прискорює розробку і тестування систем, бо не потрібно писати свою програму логічного виведення.

В Prolog речення бувають трьох видів: факти, правила, питання. У математичних термінах Prolog-програма інтерпретується так:

- факти і правила – множина аксіом;
- питання користувача – теорема;
- механізм виведення намагається з аксіом логічно довести теорему (в Prolog ця теорема називається метою).

Факт – це речення, яке фіксує (визначає) певні відносини між об’єктами. Факт є безумовно істинним твердженням. Об’єкти називаються термами або атомами. Факт може складатись з одного або кількох термів. Кількість термів називається арністю. Термами факту можуть бути константа, змінна або складений об’єкт (список або функція). Якщо об’єкти мають причинно наслідковий зв’язок, то, як правило першим подається об’єкт, що є причиною,

але це не обов’язково. Важливо дотримуватись вибраної послідовності термів факту на протязі всієї програми.

Наприклад, факт, що Наталя є матір’ю Даші має арність 2 і може бути записаний таким чином:

mother('Наталя', 'Софія').

Терми факту зв’язуються логічними операторами: **І** (кон’юнкція**),** позначається символом **« , », АБО** (диз’юнкція), позначається символом **« ; ».**

Правило – речення, що складається з заголовка і тіла. Воно може бути істинним або хибним (не істинним). Істинність залежить від істинності однієї або декількох формул, зазначених в тілі. Зазвичай правило містить кілька фактів, які повинні бути істинними для того, щоб саме правило було істинним. Правило має ім’я, після якого йдуть символи **« :- »** – ознака правила. Загальний вид правила:

B:- A_1, ..., A_n.

B називається ім’ям або заголовком правила, а **A_1, ..., A_n** – тілом. У цьому реченні ключовою логічною операцією є перевернута імплікація **B:-A** еквівалентно **B ← A**, «В випливає з А»). Символ " **:-** " означає "випливає з», символ « **,** » – кон’юнкція (логічне **І**). При необхідності застосування диз’юнкції (логічне **АБО**) використовується символ « **;** », він діє до наступної диз’юнкції, закінчення правила або закриваючої круглої дужки, наприклад**, D:- A, (B; C)**.

Наприклад, визначимо правила для родинних відносин бабуся→онуки. Відомо, що бабуся людини – це матір її матері або матір її батька. Відповідні правила виглядатимуть так:

grandmother(X, Y):- mother(X, Z), mother(Z, Y).

grandmother(X, Y): - mother(X, Z), father(Z, Y).

Ці правила можна записати одним правилом:

grandmother(X, Y):- mother(X, Z), (mother(Z, Y); father(Z, Y)).

В результаті виконання правила змінна **X –** отримає значення **ім’я бабусі,** змінна **Z** отримає значення **ім’я матері** або **ім’я батька**, змінна **Y** отримає значення **ім’я дитини з** фактів **mother** і **father** бази фактів.

Запит (питання, мета) – речення, що складається тільки з тіла. Запити позначаються символами **« ?- ».**

Запити використовують для з’ясування здійсненності деяких відносин між описаними в програмі об’єктами. Автоматична система логічного виведення Prolog розглядає питання як мету, яку треба довести. Відповідь на питання може виявитися істинною (true) або хибною (false), в залежності від того, чи може бути досягнута мета.

Програма може містити питання в тілі програми (внутрішня мета). Якщо внутрішньої мети в програмі немає, то після запуску програми система видає запрошення **« ?- »** вводити питання в діалоговому режимі (зовнішня мета). Якщо мета досягнута, система відповідає «true» («yes»), в іншому випадку «false» («no»). Слід зазначити, що відповідь «false» на питання не завжди означає хибність. Система може дати таку відповідь і в тому випадку, коли у неї просто недостатньо інформації, щоб позитивно відповісти на питання. Тобто Prolog заснований на так званій «Моделі закритого світу», в якій все, що

можна отримати на основі опису моделі є істиною, а решта – хибність. Наприклад, ми запитуємо про факт, якого нема в програмі, і отримуємо «false». Але це не означає, що в реальному житті (предметній області) такий факт не існує.

1.2. Обмеження мови Prolog.

Програма в Prolog є послідовністю зазначених вище трьох типів предикатів, які повинні закінчуватись крапкою. В тілі трьох основних предикатів можуть бути присутні інші предикати, зв'язані логічними операторами І, АБО. При запису предикатів не повинно бути зайвих пробілів. Пробіли допустимі тільки між логічними операторами.

Ім'я предикату в Prolog повинне складатись з літер латинського алфавіту і починатися з малої літери. Ім'я терму повинне починатись з малої літери латинського алфавіту, або поміщатись в одинарні лапки, якщо воно починається з великої літери або використовуються не латинські літери, наприклад кирилиця: **'Наталя'.**

Змінні. Ім'я змінної в Prolog складається з літер латинського алфавіту, цифр, знаків підкреслення і повинно починатися з великої літери або символу підкреслення. Наприклад, у реченні

grandmother(X, Y):- mother(X, Z), mother(Z, Y).

імена змінних починаються з великої латинської літери. Змінні в тілі правила еквівалентні об'єктам предметної області. Змінні можуть бути вільними або зв'язаними.

Вільна змінна – змінна, яка ще не отримала значення під час виконання програми. Вона не дорівнює ні нулю, ні пробілу; у неї взагалі немає ніякого значення. Такі змінні ще називають неконкретизованими.

Зв'язана змінна – змінна яка отримала якесь значення. Такій змінній не може бути присвоєно нове значення.

Областю дії змінної в Prolog є одне речення. У різних реченнях може використовуватися одне й теж ім'я змінної для позначення різних об'єктів. Винятком з правила визначення області дії є анонімна змінна, яка позначається символом підкреслення « _ ». Анонімна змінна наказує інтерпретатору (компілятору) Prolog проігнорувати значення терму. Анонімні змінні можуть записуватися тільки в якості терму предиката. Використовувати їх у виразах (наприклад, арифметичних) не можна.

Таким чином, програма на Prolog складається з фактів і правил, що представляють деякі знання про предметну область. Результатом роботи програми є істинність питання, яке нас цікавить. Якщо питання не містить змінних, то обчислення його значення дає відповідь «true» при його істинності, або відповідь «false» при його хибності. Якщо в питанні є змінні, то відшукуються їхні значення, при яких цей предикат і всі предикати програми стають істинними. В цьому і полягає суть логічного виведення програми на Prolog.

1.3. Приклад бази фактів і бази знань в Prolog

Розглянемо приклад бази фактів і бази знань сімейних відносин. Зауважимо, що символ % означає, що після нього йде коментар, який не впливає на програму.

```
% База фактів сімейних відносин
mother('Наталя','Софія').
mother('Софія','Марія').
mother('Наталя','Василь').
father('Василь','Марія').
% База правил (знань) експертної системи сімейних відносин
grandmother(X, Y):-
mother(X, Z),(mother(Z, Y);father(Z, Y)).
grandfather(X, Y):-
father(X, Z), (mother(Z, Y); father(Z, Y)).
```

Розглянемо, як питання природною мовою записуються в Prolog.

Питання 1. Чи є Наталя матір'ю Софії?

```
?-mother('Наталя', 'Софія').
true                          % Відповідь системи.
```

Відповідь означає, що такий факт в базі фактів існує (є істинним), тобто Наталя є матір'ю Софії.

Питання 2. Хто є матір'ю Софії?

Для таких питань необхідно використовувати змінні, наприклад, змінну з ім'ям X. Змінна X отримує значення першого терму предикатів-фактів **mother**, в яких другим термом є терм **'Софія'**

```
?-mother(X, 'Софія').
X = 'Наталя'
```

Рядок повідомлення **X = 'Наталя'** означає, що відповідь знайдена і матір'ю Софії є Наталя.

Питання 3 – Чи є у Софії матір? Використовується анонімна змінна.

```
?-mother(_, 'Софія').
true                          % Відповідь системи.
```

Питання 4 – Знайти в базі фактів всіх матерів і їхніх дітей.

```
?-mother(X, Y).
X = 'Наталя',
Y = 'Софія'
X = 'Софія',
Y = 'Марія'
X = 'Наталя',
Y = 'Василь'
```

Питання 5 – Для кого Наталя є бабусею?

```
?-grandmother('Наталя', X).
X = 'Марія'
X = 'Марія'
```

В даному випадку отримуємо дві однакові відповіді «Марія», тому що правило grandmother в одному випадку спрацювало по ланцюжку

«Наталя-Софія-Марія», а в іншому – «Наталя-Василь-Марія». Очевидно, що в наведеній базі знань бабусі з ім'ям **'Наталя'** – це дві різні жінки з однаковими іменами і онучки теж мають однакові імена **'Марія'**.

1.4. Процедура виведення в Prolog

При пошуку розв'язку (доказу мети) в Prolog використовується метод перебору з поверненнями (з пошуком в глибину). Prolog при доказі твердження по черзі намагається встановити істинність предикатів (тверджень), що входять в нього. Якщо перший предикат істинний, то Prolog переходить до другого. Якщо і він істинний, то переходить до третього. Якщо другий предикат хибний, то Prolog намагається встановити його істинність при інших значеннях змінних, що входять в нього. Якщо цього не вдається зробити, то він повертається до першого предикату і намагається встановити його істинність для нових значень змінних, а потім знову повертається до доказу другого предиката. Така процедура повторюється до тих пір, поки не буде досягнута істинність останнього предиката. Після доведення істинності останнього предиката мети (запиту) Prolog завершує роботу. Процес повернення при доказі твердження в Prolog називається **backtracking.**

Розглянемо дії Prolog при роботі з питанням 5. Процедура виведення (перебору з поверненнями) для цього прикладу наведена в табл. 2.

Таблиця 1.2 – Приклад виведення в SWI-Prolog

№ з/п	Предикат запиту	Предикат бази знань, що перевіряється	Результат
1	**mother('Наталя', Y)**	**mother('Наталя', 'Софія')**	**Y = 'Софія'**
2	mother(Y, Z) ≡ mother('Софія', Z)	mother('Наталя', 'Софія')	backtracking
3	**mother(Y, Z) ≡ mother('Софія', Z)**	**mother('Софія', 'Марія')**	**Z = 'Марія'**
5	mother(Y, Z) ≡ mother('Софія', Z)	mother('Наталя', 'Василь')	backtracking
6	father(Y, Z) ≡ father('Софія', Z)	father('Василь, 'Марія')	backtracking
7	mother('Наталя', Y)	mother('Софія', 'Марія')	backtracking
8	**mother('Наталя', Y)**	**mother('Наталя','Василь')**	**Y = 'Василь'**
9	mother(Y, Z) ≡ mother('Василь', Z)	mother('Наталя', 'Софія')	backtracking
10	mother(Y, Z) ≡ mother ('Василь', Z)	mother('Софія', 'Марія')	backtracking
11	mother(Y, Z) ≡ mother ('Василь', Z)	mother('Наталя', 'Василь')	backtracking
12	**father(Y, Z) ≡ father('Вася', Z)**	**father('Василь', 'Марія')**	**Z = 'Марія'**

Примітка. Жирним виділені рядки, для яких значення предиката є істинним.

1.5. Програмний засіб SWI-Prolog

SWI-Prolog є вільно розповсюджуваною (Free Software) реалізацією (діалектом) мови програмування Prolog, яка практично повністю відповідає стандарту ISO/ EC13211.

SWI-Prolog дозволяє розробляти програми будь-якої спрямованості, включаючи Web-додатки і паралельні обчислення, але основним напрямом використання є розробка експертних систем, навчальних програм, інтелектуальних ігор тощо.

Програмування в SWI-Prolog (http://www.swi-prolog.org) можливо в різних варіантах рис. 1.1:

- за допомогою стандартного offline-середовища у вільному доступі, можна завантажити з сайту і встановити (кнопка **Download SWI-Prolog);**

- за допомогою online-середовища на сайті http://www.swi-prolog.org (кнопка **Try SWI-Prolog online).**

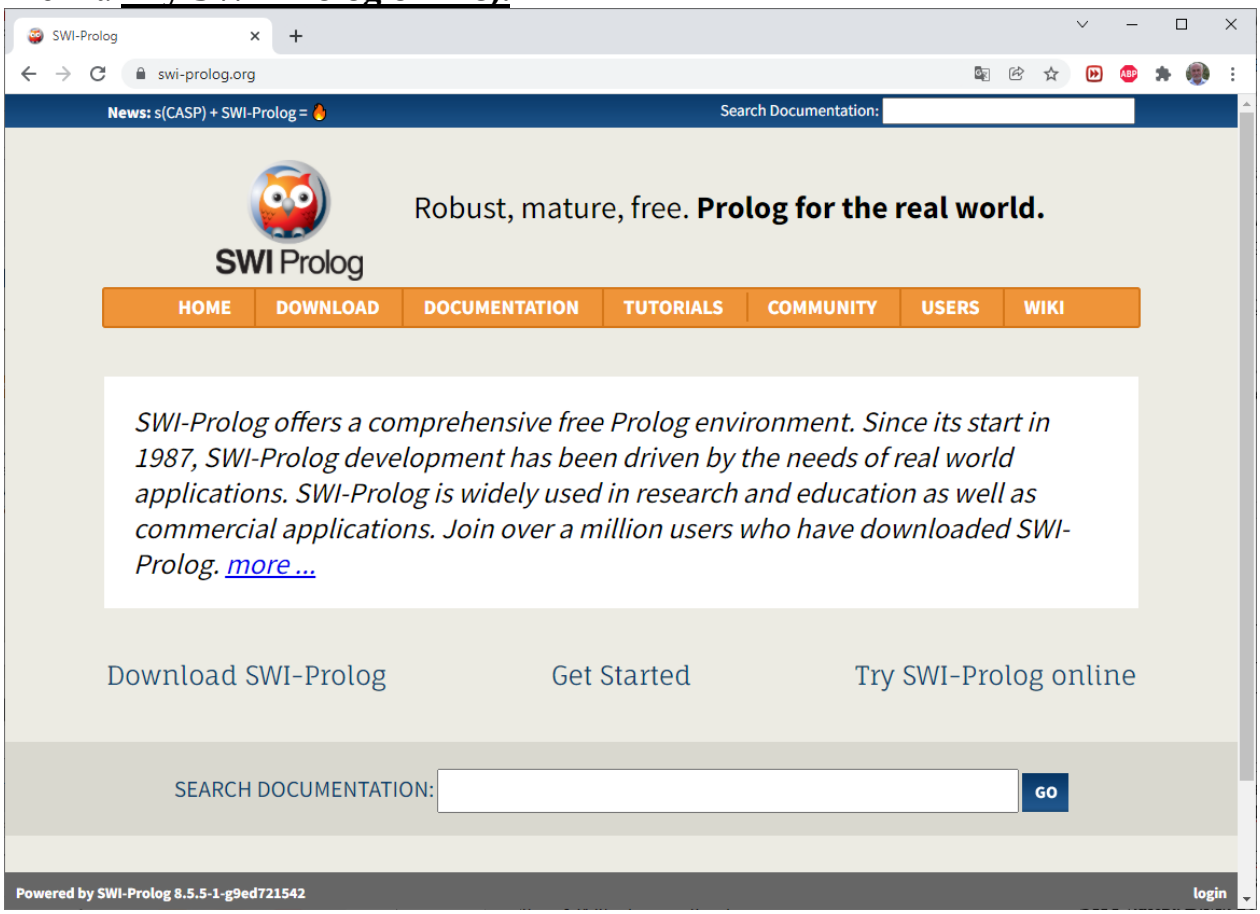


Рисунок 1.1 – Початок роботи з SWI-Prolog

Для переходу в режим редагування і виконання програм необхідно натиснути на кнопку «Program» (рис. 1.2). Після цього можна створювати і редагувати Prolog-програму, створювати запити і отримувати відповіді на запити (рис. 1.3).

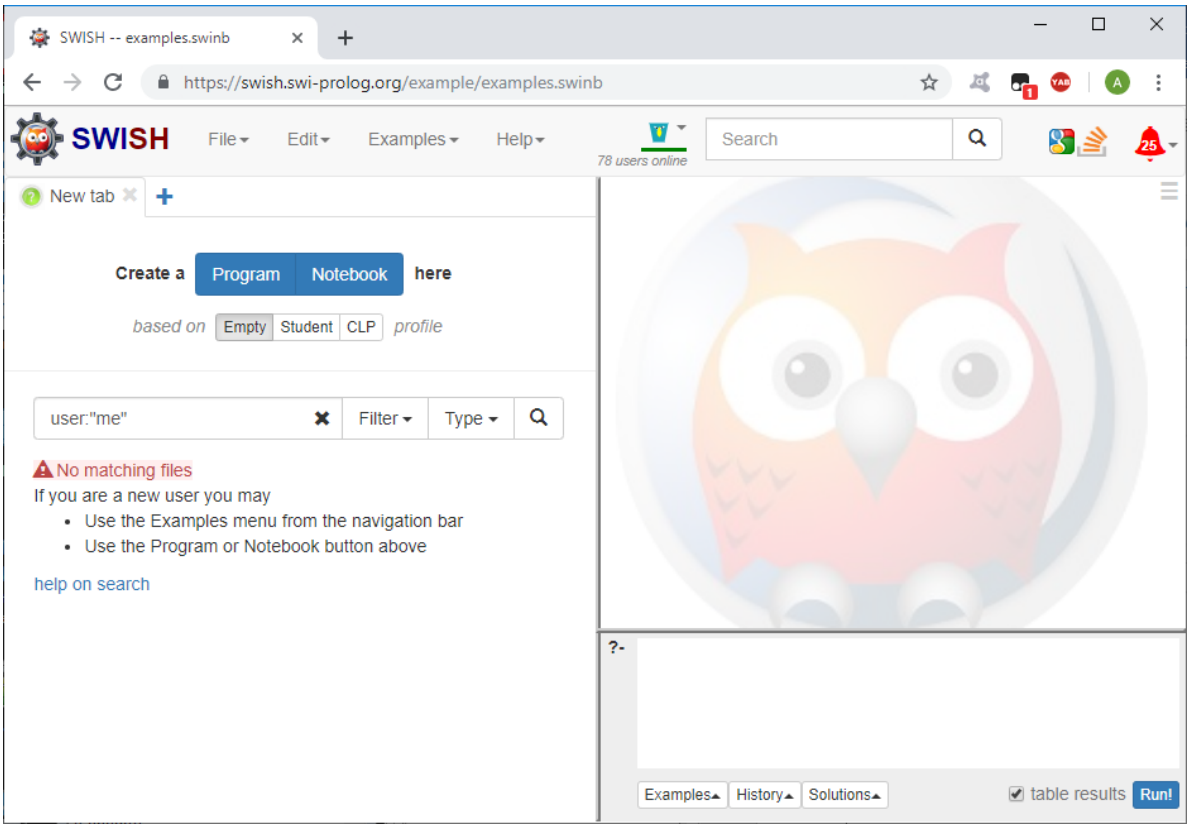


Рисунок 1.2 – Стандартне online-середовище програмування SWI-Prolog

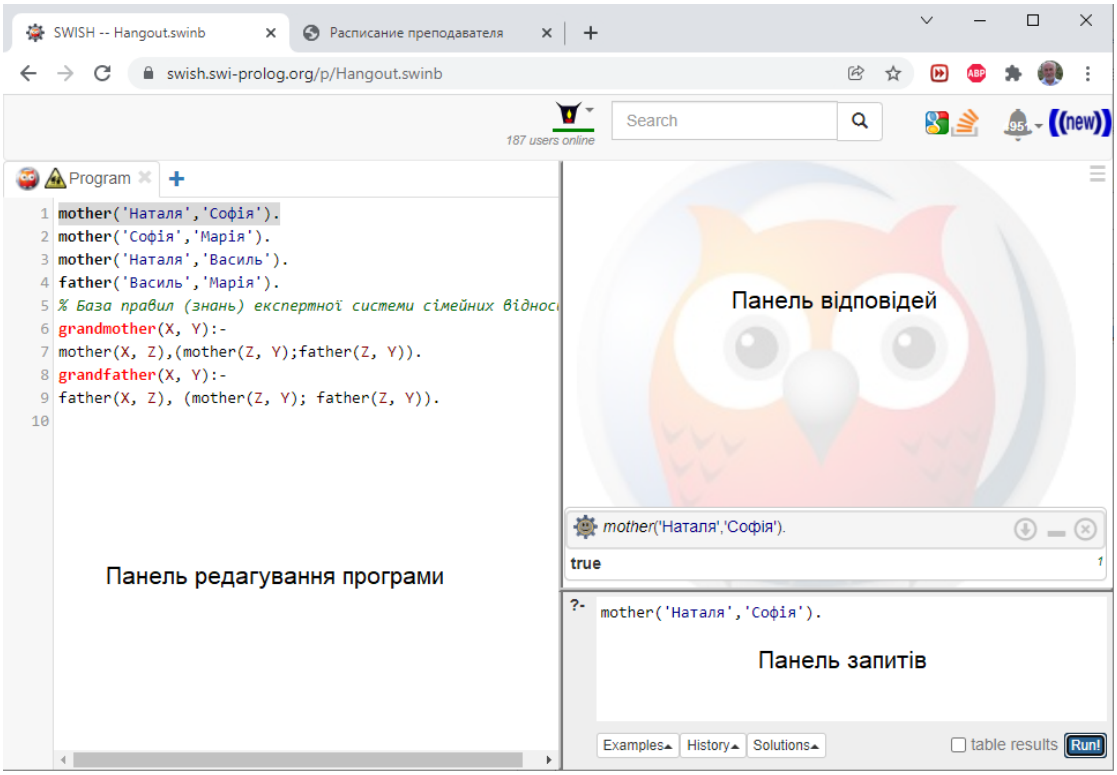


Рисунок 1.3 – Режим редагування і виконання програм

У лівій панелі здійснюється редагування програми, яка містить факти і правила.

У правій нижній панелі здійснюється введення запитів (питань) і запуск їх на виконання за допомогою кнопки «Run!».

У правій верхній панелі інтерпретатор SWI-Prolog виводить результат запиту. У разі якщо відповідей на запит буде більше ніж одна, всі інші можна вивести за допомогою кнопок «Next», «10», «100» і «1,000». При необхідності виведення можна зупинити кнопкою STOP.

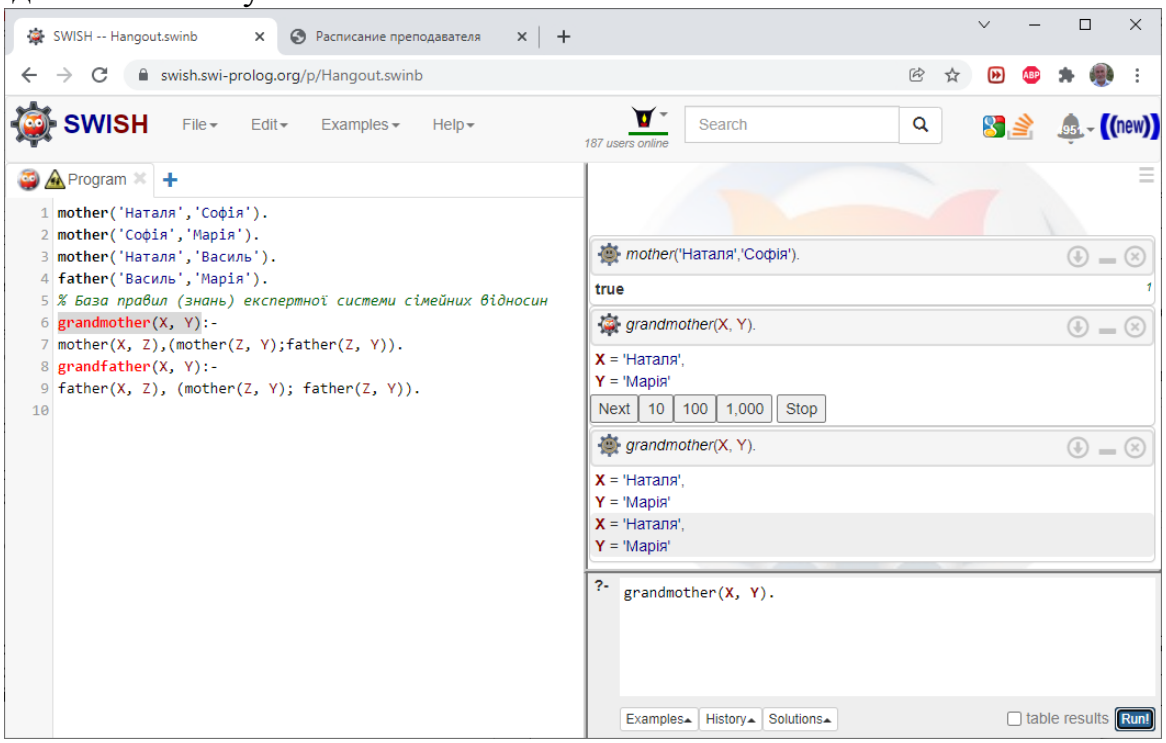


Рисунок 1.4 – Застосування кнопок «Next», «10», «100» і «1,000».

Практична робота 2
Створення експертної системи засобами SWI-Prolog.

Мета. Вивчити склад і основні принципи побудови експертних систем на засадах логічних моделей подання знань та навчитися будувати інтерфейс користувача експертної системи.

Завдання

1. В якості предметної області обрати предметну область згідно з варіантом (табл. 1).
2. Визначити мету, задачі експертної системи та питання користувача до експертної системи.
3. Визначити правила бази знань.
4. Побудувати блок-схему (дерево) логічних можливостей для вибору.
5. Побудувати схему структуру діалогу з користувачем.
6. Створити базу знань за допомогою засобів SWI-Prolog.
7. Проаналізувати отримані результати і зробити висновки.

Примітка. Номер варіанта співпадає з номером зі списку групи. Якщо номер зі списку перевищує кількість варіантів, варіант визначається як ціле число залишку від ділення номера зі списку на кількість варіантів. Якщо номер зі списку кратний кількості варіантів, то обирається останній варіант. Приклад: номер у списку – 31, варіантів 20 то варіант – 11 (31/20 =1 залишок 11).
номер у списку – 20, варіантів 20 то варіант – 1

Таблиця 2.1 – Перелік предметних областей для експертної системи.

№ з/п	Предметна область
	Індивідуальний підбір косметики
	Індивідуальний підбір одягу
	Індивідуальний підбір взуття
	Індивідуальний підбір мобільного телефону
	Індивідуальний підбір музичного центру
	Індивідуальний підбір відеокамери
	Індивідуальний підбір телевізора
	Індивідуальний підбір комп’ютера
	Індивідуальний підбір автомобіля
	Індивідуальний підбір житла
	Індивідуальний підбір спеціальності для абітурієнтів
	Класифікація птахів
	Класифікація квітів
	Класифікація риб
	Класифікація дерев
	Класифікація овочів
	Класифікація собак
	Класифікація мавп
	Класифікація фруктів
	Класифікація комах

Зміст звіту

1. Титульна сторінка.
2. Тема практичного заняття.
3. Мета заняття.
4. Завдання.
5. Опис виконання завдань по пунктам з наданням блок-схем, рисунків і скріншотів.
6. Текст SWI-Prolog програми
7. Висновки.

Контрольні питання

1. Структура продукційної експертної системи.
2. Етапи створення експертної системи.
3. Режими роботи експертних систем.
4. Організація вводу-виводу даних в Prolog.
5. Динамічне додавання факту в Prolog.

Теоретичні відомості.

2.1. Подання знань в продукційних системах

Продукція визначається як четвірка:

$$< (i)\ Q;\ P;\ A \Rightarrow B;\ N >,$$

де:

(i) – ім’я продукції;

Q – предметна область (сфера застосування);

P – умова застосування продукції;

$A \Rightarrow B$ – ядро продукції (правило) інтерпретується як «**Якщо маєш А, зроби В**» або «**Якщо А то В**»;

N – післяумова продукції (процедури, які необхідно виконати після реалізації ядра.

Найбільше застосування ця модель знайшла в продукційних (логічних) експертних системах (рис. 2.1). Для спрощення реалізації таких систем створена мова програмування Prolog. Її особливістю є те, що алгоритм логічного виведення вбудований в неї, що не вимагає писати програму логічного виведення, що прискорює розробку і тестування систем.

У математичних термінах Prolog-програма інтерпретується наступним чином:

- факти і правила – множина аксіом;
- питання користувача – теорема;
- механізм виведення намагається з аксіом логічно довести цю теорему (в Prolog ця теорема називається метою).

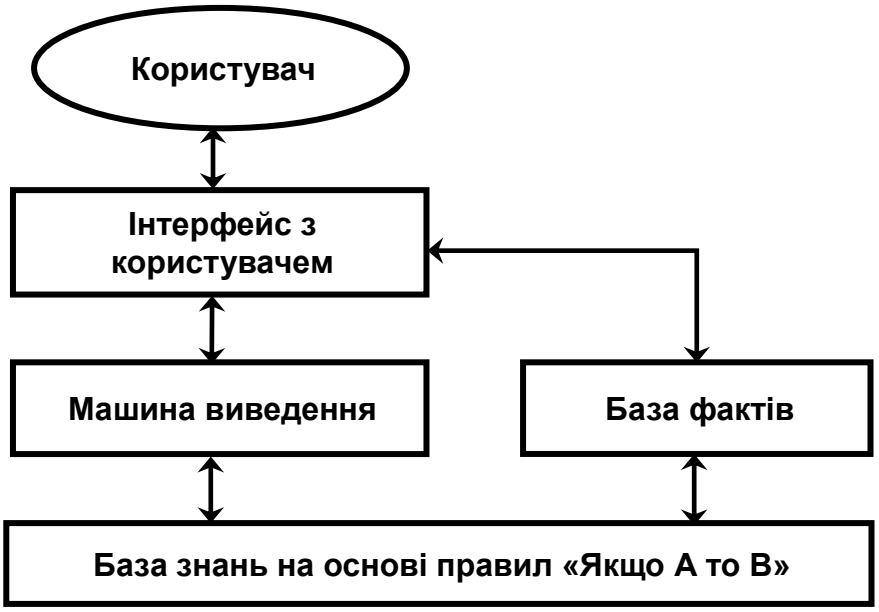


Рисунок 2.1 – Структура продукційної експертної системи

Факти і правила використовуються для доведення мети. Prolog шукає факти і заголовки правил і зіставляє з метою. Факт із бази фактів і факт з умови правила відповідають один одному, якщо виконуються наступні три умови: імена фактів однакові; факти мають рівну кількість термів; терми розташовані в однакових позиціях.

2.2. Етапи створення експертної системи.

Ідентифікація. Необхідно визначити мету і задачі експертної системи; визначити експертів і тип користувачів.

Концептуалізація. Проводиться змістовний аналіз предметної області; виділяються основні поняття і їх взаємозв’язки; визначаються методи рішення задач.

Формалізація. Вибираються програмні засоби розробки ЕС, визначаються способи представлення усіх видів знань, формалізуються основні поняття.

Виконання. Здійснюється наповнення бази знань. Знання "витягаються" з експерта, представляються у логічному або інших видах для реалізації програмно.

Тестування. Експерт і інженер по знанням з використанням діалогових і пояснювальних засобів перевіряють компетентність ЕС. Процес тестування продовжується до визнання експертом необхідного рівня компетентності системи.

Дослідна експлуатація. Перевіряється придатність ЕС для кінцевих користувачів**.**

2.3. Режими роботи експертних систем

Експертні системи працюють у двох режимах:

- придбання знань;
- режим рішення задачі (режим консультації).

У режимі придбання знань інженер по знанням по результатам спілкування з експертом наповнює ЕС новими знаннями і фактами.

У режимі консультації користувач взаємодіє з ЕС для рішення своєї задачі шляхом діалогу. Якщо в системі не вистачає фактів для доведення мети, наприклад встановлення діагнозу, то вона задає питання користувачеві і той ці факти надає. Після закінчення діалогу машина виведення доводить або спростовує мету.

2.4. Модель експертної системи для вибору туристичної поїздки

(Приклад розробки)

Етап ідентифікації експертної системи.

Постановка завдання: у процесі вибору туру важко зупинитися на якійсь моделі, тому що велика кількість фірм надає подібні послуги з різними характеристиками, і в цьому достатньо важко орієнтуватися. Пропонується розробити прототип експертної системи (ЕС) прийняття рішень по вибору туру за перевагами, які є важливими для клієнта (користувача).

Призначення ЕС: консультування клієнта під час визначення оптимального туру за властивостями, яким він надає перевагу.

Сфера застосування прототипу ЕС: туристичні фірми, що займаються продажом турів.

Ціль: вибір оптимального варіанта туру для клієнта згідно з його потреб і вимог до послуги.

Вхідні дані: країна відвідування, вид відпочинку, тривалість відпочинку.

Очікувані результати: конкретний тур.

Об’єкти (фактори) Предметної Області (ПрО). Тур – це послуга, на вибір якої впливають різні фактори. Фактори, наприклад, визначені з соціологічного опитування і ними є наступні:

- країна відвідування;
- вид відпочинку;
- тривалість відпочинку.

Етап концептуалізації експертної системи.

У розпорядженні туристичної фірми є можливість направляти в три країни (Єгипет, Францію, Норвегію), з трьома типами відпочинку (В Горах, На курорті, Екскурсії) і з двома термінами відпочинку (Тиждень, На вихідні)

Загальна кількість можливих варіантів турів дорівнює добутку всіх значень атрибутів (3 х 3 х 2 = 18). Але серед цих варіантів експерт відібрав тільки 6, які відповідають існуючим на даний час пропозиціям. Дерево логічних можливостей вибору туру подане на рис. 2.1.

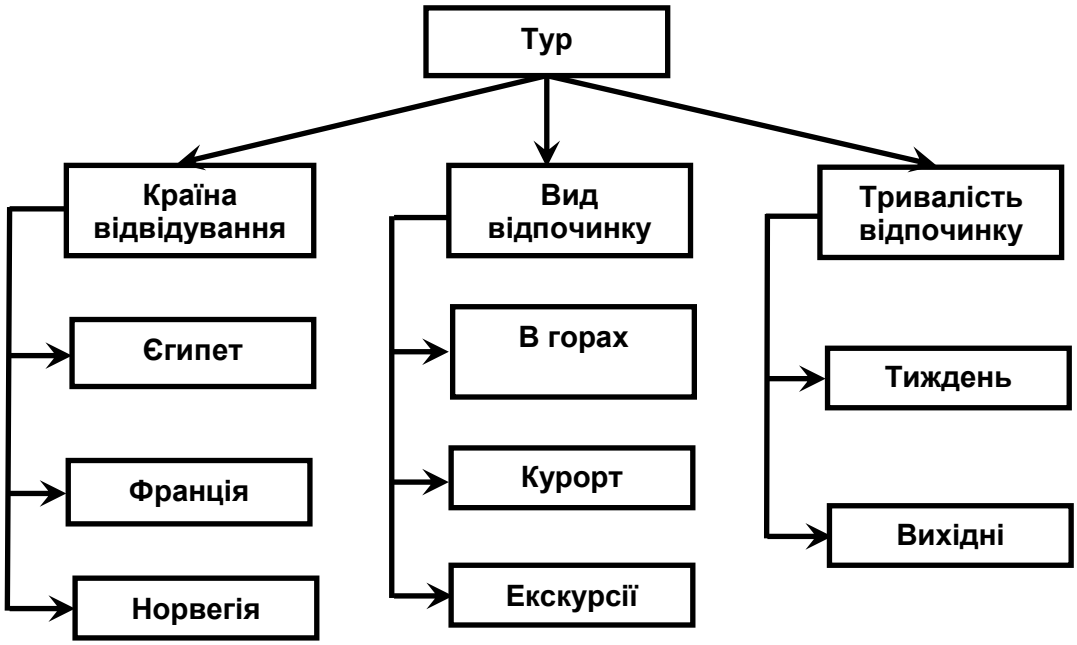


Рисунок 2.1 – Дерево логічних можливостей для вибору туру

Дану структуру можна представити і у виді таблиці (табл. 2.), де імена фактів-предикатів відповідають вимогам Prolog.

Таблиця 2 – Атрибути Бази знань

Атрибут (термін)	Питання (опис)	Відповідь
Країна	Яку країну Ви бажаєте відвідати?	egypt
		france
		norway
Тип відпочинку	Якому виду відпочинку Ви надаєте перевагу?	mountain
		resort
		sightseeing
Тривалість відпочинку	Як довго Ви хочете відпочивати?	weekend
		week

Загальна структура правил має вид: **ЯКЩО** (A і B і C) **ТО** Тур,
де A – Країна (country), B – Вид Відпочинку (type), C – Тривалість відпочинку (duration); Тур (tour) – назва туру.

Приклад правила:

ЯКЩО (country = egypt, type = mountain, duration = weekend)

ТО tour = 'Квадро-тур на Синай'.

2.5. Програмна реалізація експертної системи з вибору туру

На початку діалогу в базі фактів факти відсутні. Під час діалогу користувач вводить факти і після цього робиться логічне виведення. Для організації діалогу застосовуються вбудовані предикати **write(A)** – вивід на екран, **nl –** перехід на новий рядок на екрані, **read(A)** – читання з клавіатури, **assert(A)** – динамічне додавання факту в базу фактів (дозвіл на зміну бази фактів дає предикат **:- dynamic**. Система задає три питання: **ask1, ask2, ask3**.

```
main:-
greeting,
ask1, ask2, ask3,
find_tour(Product),
describe(Product), nl.
greeting:-
write('Експертна система з підбору туристичної путівки'), nl, nl.
```

```
:- dynamic(country/1).
:- dynamic(type/1).
:- dynamic(duration/1).
ask1:- write('Яку країну Ви бажаєте відвідати?)', nl,
write (egypt, france, norway), nl,
  read(N), assert(country(N)).
ask2:- write('Якому виду відпочинку Ви надаєте перевагу?'), nl,
write(mountain, resort, sightseeing), nl,
  read(N), assert(type(N)).
ask3:-write('Як довго Ви хочете відпочивати?'), nl,
write(weekend, week), nl,
   read(N), assert(duration(N)).
find_tour(Tour) :-
tour(Tour), !.
            %База знань (правил)
tour('Квадро-тур на Синай') :-
  country(egypt), type(mountain), duration(weekend).
tour('Піший похід на гору Мойсея') :-
   country(egypt), type(mountain), duration(week).
tour('Вікенд в Хургаді') :-
   country(egypt), type(resort), duration(weekend).
tour('Тиждень в Шарль-Ем-Шейху') :-
   country(egypt), type(resort), duration(week).
tour('Похід на Монт Пелат') :-
   country(france), type(mountain), duration(weekend).
tour('Екскурсійний тур по Осло') :-
   country(norway), type(sightseeing), duration(weekend).
        % Вивід результату коли яке-небудь правило стане істинним (спрацює)
describe(Tour):-
write('Тур, що Вам підходить: '), nl,
   write(Tour), nl, nl.
```

Для запуску програми необхідно на панелі запитів SWI-Prolog набрати **main** і натиснути **Run!**

Додаток А

Операції та вбудовані предикати SWI-Prolog

Таблиця 2.3 – Деякі операції та предикати SWI-Prolog корисні для експертних систем.

Операція / Предикат	Призначення
=	Для змінної, що стоїть зліва від операції: • вільної – присвоювання без обчислення виразу праворуч від операції; • зв'язаної – порівняння без обчислення виразу праворуч від операції.
<, =<, >=, >	Арифметичні (тільки для чисел) операції порівняння
=:=	арифметична рівність
=\=	арифметична нерівність
is	Для змінної, що стоїть зліва від операції: • вільної – присвоювання з обчисленням виразу праворуч is; • зв'язаної – порівняння з обчисленням виразу праворуч від is.
@<, @=<, @>=, @>	Операції порівняння для констант і змінних будь-якого типу (чисел, рядків, списків і т.д.)
==	Рівність констант і змінних будь-якого типу
\==	Нерівність констант і змінних будь-якого типу
not(A)	Заперечення логічного виразу А
read(A)	Читання значення з клавіатури і присвоювання його змінній А
write(A)	Друк А на екрані з установкою курсору після останнього надрукованого символу
writeln(A)	Друк А на екрані з установкою курсору в початок наступного рядка
nl	Установка курсора в початок наступного рядка
!	Предикат (cut, скоротити) забороняє повернення (backtracking) далі тієї точки, де він стоїть
assert(A), assertz(A)	Динамічне додавання факту (правила) в початок списку подібних фактів (правил) бази фактів і знань
asserta(A)	Динамічне додавання факту (правила) в кінець списку подібних фактів (правил) бази фактів і знань
retract(A)	Видалення першого факту (правила) бази фактів і знань
retractall(A)	Видалення всіх фактів (правил) бази фактів і знань з ім'ям А
dynamic(A/n)	Вказівка, що предикат А може змінюватись в процесі виконання програми за допомогою предикатів **assert і retract**, n – кількість термів предикату А

Практична робота 3
Створення системи нечіткого керування засобами MATLAB

Мета. Вивчення основ створення систем нечіткого виведення і керування, дослідження алгоритму Мамдані, набуття практичних навичок роботи з пакетом Fuzzy Logic Toolbox.

Завдання

1. Для предметної області згідно з варіантом (таблиця 3.1):
   - зробити короткий опис предметної області, визначити вхідні і вихідні змінні;
   - сформулювати мету функціонування;
   - визначити діапазони значень вхідних і вихідних змінних;
   - записати правила функціонування системи.
2. Засобами Fuzzy Logic Toolbox створити систему нечіткого керування (функції належності вхідних і вихідних змінних, базу нечітких правил).
3. Результати зберегти у виді файлу MATLAB (*.fis).
4. Дослідити створену систему нечіткого виведення на різних значеннях вхідних даних.
5. Проаналізувати отримані результати і зробити висновки.

Таблиця 3.1 – Перелік рекомендованих тем для створення систем.

№ з/п	Тема	Вхід / Кількість термів	Вихід / Кількість термів
1.	Керування душем	Температура води / 3	Кут повороту крану гарячої води / 3
2.	Керування електричним двигуном	Швидкість обертання / 5	Напруга / 5
3.	Керування системою опалення.	Температура / 3 Вік людини / 5	Клапан / 3
4.	Визначення рангу студента	Середня успішність / 3 Термін виконання / 3	Ранг / 3
5.	Визначення суми кредиту	Кількість наданих кредитів / 3 Розмір попередніх кредитів / 3	Сума кредиту / 3
6.	Керування клапаном для води	Рівень води / 3	Клапан / 3
7.	Керування кермом автомобіля	Відстань / 3 Положення перешкоди відносно напряму руху / 3	Поворот керма / 5
8.	Конкурента здатність	Кількість втрачених клієнтів / 3 Кількість нових клієнтів / 3	Прибуток / 3
9.	Керування опаленням	Температура / 3	Кут повороту крану / 3
10.	Керування кондиціонером	Температура / 3	Клапан / 3

Примітка. Номер варіанта співпадає з номером зі списку групи. Якщо номер зі списку перевищує кількість варіантів, варіант визначається як у попередніх роботах.

Зміст звіту

1. Титульна сторінка з
2. Тема практичного заняття.
3. Мета заняття.
4. Номер варіанту та завдання до роботи.
5. Опис виконання завдань по пунктам з наданням скріншотів.
6. Текст створеного *.fis файлу.
7. Висновки.

Контрольні питання

1. Структура нечіткого логічного виведення
2. Типи функцій належності.
3. Метод нечіткого виведення Мамдані
4. Методи дефазіфікації для нечіткого виведення Мамдані.

Теоретичні відомості

3.1. Принципи нечіткого логічного виведення

Нечітке логічне виведення (рис. 3.1) полягає в тому, що від точних (наприклад, виміряних фізичними приладами) вхідних і вихідних змінних ми переходимо до неточних, заданих функція належності. Зв'язок між вхідними і вихідними функція належності задається через нечіткі правила. Машиною нечіткого логічного виведення на основі цих правил створюється результуюча нечітка вихідна функція належності, з якої за одним з методів дефазифікації визначається чітке значення, яке може бути конкретним чисельним висновком або у разі системи керування керуючим впливом на фізичний об'єкт.

Рисунок 3.1 – Нечітке логічне виведення

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

Припустимо, що базу знань утворюють два нечітких правила:

П1: Якщо $x_1 \in A_1^1$ і $x_2 \in A_1^2$, То $y \in B^1$;

П2: Якщо $x_1 \in A_2^1$ і $x_2 \in A_2^2$, То $y \in B^2$,

де: x_1 , x_2 – імена вхідних змінних, y – ім'я вихідної змінної,
A_1^1 , A_1^2, A_2^1, A_2^2 – функції належності вхідних змінних,
B^1, B^2 – функції належності вихідних змінних.

На прикладі цих правил **алгоритм нечіткого виведення Мамдані** виглядає наступним чином (рис. 3.2):

1. Введення нечіткості (fuzzification): для заданих (чітких) значень аргументів $x_1 = x_1(0)$, $x_2 = x_2(0)$ знаходяться степені істинності для передумов правил П1, П2.

2. Знаходиться рівні відсічення для передумов кожного з правил (з використання правила мінімуму):

$$\alpha_1 = A_1^1 [x_1(0)] \wedge A_1^2 [x_2(0)]$$
$$\alpha_2 = A_2^1 [x_1(0)] \wedge A_2^2 [x_2(0)],$$

де $\wedge$ – операція нечіткого логічного мінімуму

3. Знаходиться усічені рівні вихідних функцій належності

$$B^1[y(0)] = (\alpha_1 \wedge B^1)$$
$$B^2[y(0)] = (\alpha_2 \wedge B^2)$$

4. З використанням операції max (позначеної як "v") проводиться об’єднання знайдених усічених функцій, що призводить до отримання підсумкової нечіткої множини для змінної виходу з функцією належності:

$$B = (\alpha_1 \wedge B^1) \vee (\alpha_2 \wedge B^2)$$

5. Приведення до чіткості (для знаходження $y(0)$) проводиться, наприклад, методом центроїда (як центр ваги фігури під кривою функції належності)

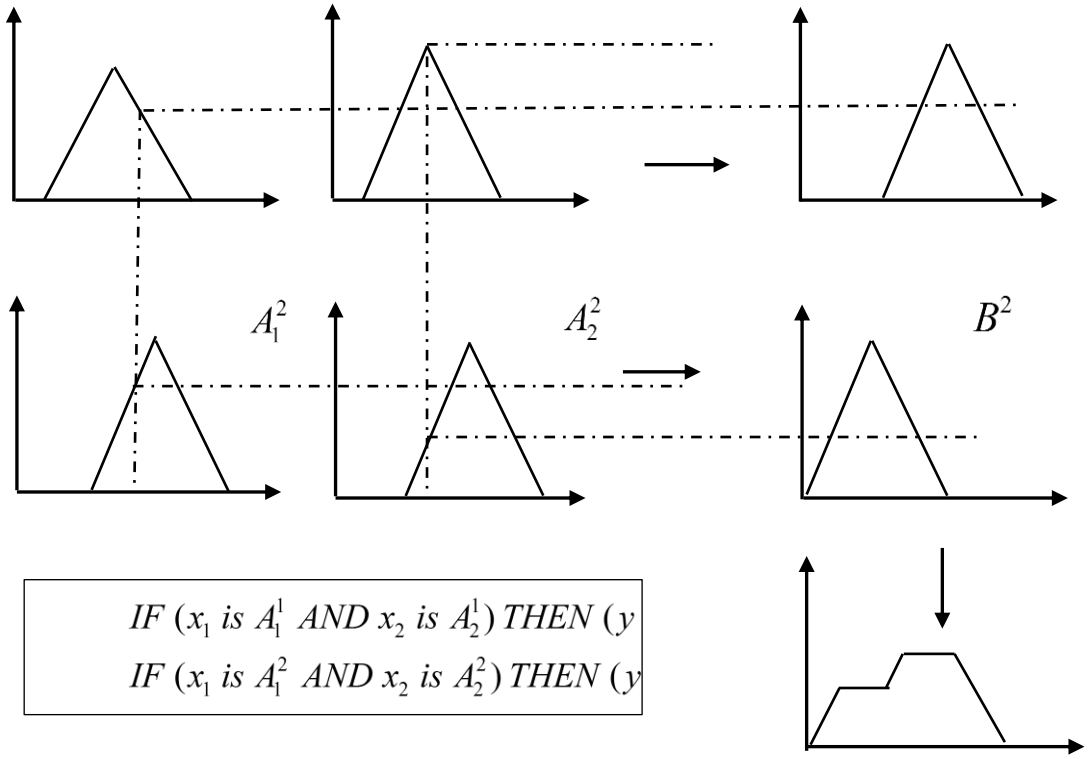


Рисунок 3.2 – Послідовність дій алгоритму Мамдані

3.2. Fuzzy Logic Toolbox

Операції з нечіткою логікою у пакеті MATLAB дозволяє виконувати модуль *Fuzzy Logic Toolbox*. Він дозволяє створювати системи нечіткого логічного виведення і нечіткої класифікації в рамках середовища MATLAB, з можливістю інтегрування в 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.

Fuzzy Logic Toolbox містить наступні категорії програмних інструментів: функції; інтерактивні модулі з графічним користувальницьким інтерфейсом (GUI); блоки для пакета Simulink; демонстраційні приклади.

FIS Editor

FIS Editor (FIS редактор) призначений для створення, збереження, завантаження і графічного друкованого подання результатів нечіткого логічного виведення, а також для редагування: типу системи; найменування системи; кількості вхідних і вихідних змінних; найменування вхідних і вихідних змінних; параметрів нечіткого логічного виведення.

Завантаження FIS Editor відбувається за допомогою команди fuzzy в робочій області MATLAB. У результаті з'являється інтерактивне графічне вікно (рис. 3.3). У нижній частині графічного вікна FIS Editor розташовані кнопки **Help** і **Close** для виклику вікна довідки і закриття редактора.

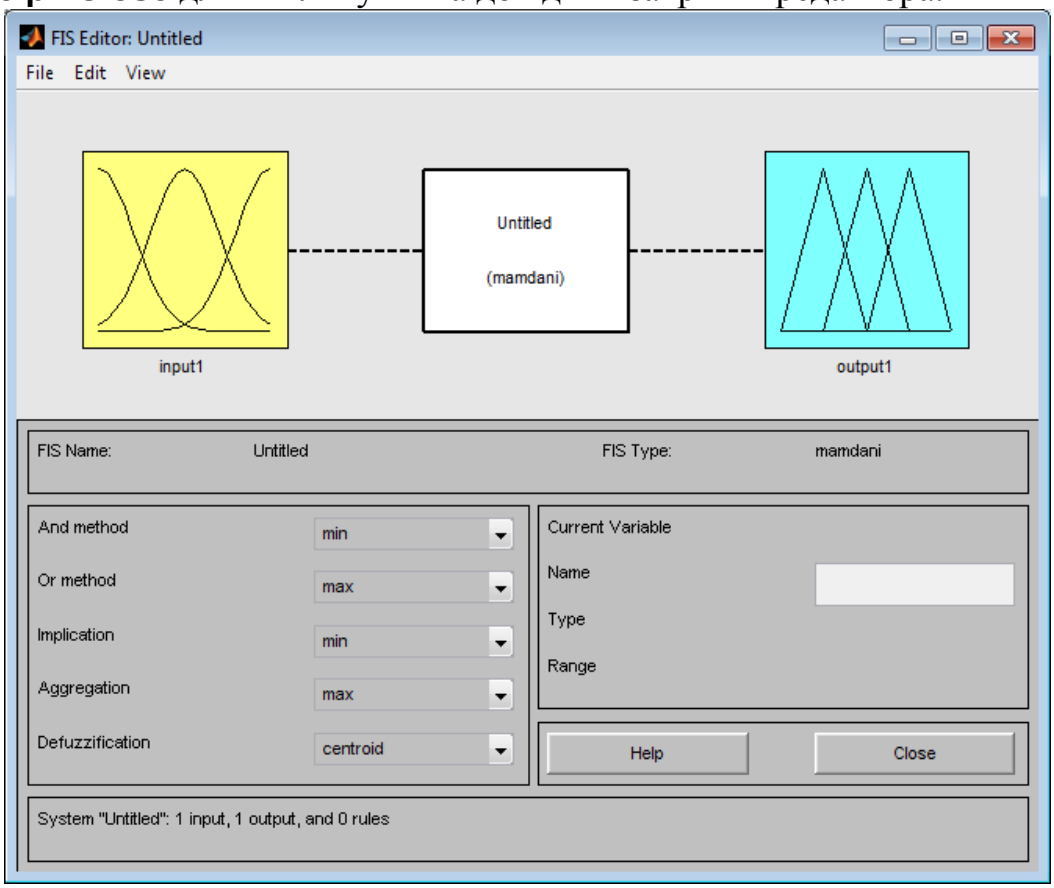


Рисунок 3.3 – Робоче вікно FIS Editor

FIS Editor містить 8 меню. Це три загальносистемних меню – **File, Edit, View,** і п’ять меню для вибору параметрів нечіткого логічного виведення – **And Method, Or Method, Implication, Aggregation і Defuzzification**.

Меню File

Це загальне меню для всіх GUI-модулів використовуваних із системами нечіткого логічного виведення. Команди меню **File.**

New FIS – створення нової систему нечіткого логічного виведення. При виборі цієї команди з’являться дві альтернативи: Mamdani (Ctrl+N) і Sugeno, що визначають тип створюваної системи. За промовчанням створюється типу Mamdani.

Import – завантаження раніше створеної систему нечіткого логічного виведення. При виборі команди з’являться дві альтернативи **From Workspace...** і **From file…**, що дозволяють завантажити систему нечіткого логічного виведення з робочої області MATLAB і з диска, відповідно. При виборі команди **From Workspace...** з’явиться діалогове вікно, у якому необхідно вказати ідентифікатор системи нечіткого логічного виведення, що знаходиться в робочій області MATLAB. При виборі команди **From file…** з’явиться діалогове вікно, у якому необхідно вказати ім’я файлу системи нечіткого логічного виведення. Файли систем нечіткого логічного виведення мають розширення **.fis** . Завантажити систему нечіткого логічного виведення з файлу можна також натисканням Ctrl+N чи командою fuzzy FIS_name, де FIS_name – ім’я файлу системи нечіткого логічного виведення.

Export – копіювання систему нечіткого логічного виведенняю При виборі команди з’являться дві альтернативи **To Workspace...** і **To file…**, що дозволяють скопіювати систему нечіткого логічного виведення в робочу область MATLAB і на диск, відповідно. При виборі команди **To Workspace...** (Ctrl+T) з’явиться діалогове вікно, у якому необхідно вказати ідентифікатор системи нечіткого логічного виведення, під яким вона буде збережена в робочій області MATLAB. При виборі команди **To file…** (Ctrl+S) з’явиться діалогове вікно, у якому необхідно вказати ім’я файлу системи нечіткого логічного виведення.

Print (Ctrl+P) **–** друк копії графічного вікна.

Close (Ctrl+W) **–** закриття графічного вікна.

Меню Edit

Undo (Ctrl+Z) – скасовує раніше зроблену дію.

Add Variable... – додавання в систему нечіткого логічного виведення ще однієї змінної. При виборі цієї команди з’являться дві альтернативи **Input** і **Output**, що дозволяють додати вхідну і вихідну змінну, відповідно.

Remove Selected Variable (Ctrl+X) – видаляє поточну змінну із системи. Ознакою поточної змінної є червона окантовка її прямокутника. Призначення поточної змінної відбувається за допомогою однократного щиглика лівої кнопки миші по її прямокутнику.

Membership Function... (Ctrl+2) – відкриває редактор функцій належності.

Rules… (Ctrl+3) – відкриває редактор бази правил (знань). Ця команда може бути також виконана натисканням Ctrl+3.

Меню View

Це загальне меню для всіх GUI-модулів, використовуваних із системами нечіткого логічного виведення. Дане меню дозволяє відкрити вікно візуалізації нечіткого логічного виведення. Команди:

Rules (Ctrl+5) – відкриття списку правил системи.

Surface (Ctrl+6) – вікно виведення поверхні “вхід-вихід”, що відповідає системі нечіткого логічного виведення (команда.

Меню And Method

Це меню встановлює реалізації логічної операції «І». Команди:

min – мінімум;

prod – множення.

Custom… – власна реалізація операції «І». Після натиснення необхідно в графічному вікні, що з’явилося, надрукувати ім’я функції, що реалізує цю операцію. Перед цим функцію треба створити як .m файл.

Меню Or Method

Це меню встановлює реалізації логічної операції "АБО". Команди:

max – максимум;

probor – імовірнісне «АБО».

Custom… – власна реалізація логічної операції «АБО». Після натиснення необхідно в графічному вікні, що з’явилося, надрукувати ім’я функції, що реалізує цю операцію. Перед цим функцію треба створити як .m файл.

Меню Implication

Це меню встановлює реалізації логічної операції імплікації. Команди:

min – мінімум;

prod – множення.

Custom… – власна реалізація логічної операції імплікації. Після натиснення необхідно в графічному вікні, що з’явилося, надрукувати ім’я функції, що реалізує цю операцію. Перед цим функцію треба створити як .m файл.

Меню Aggregation

Це меню встановлює реалізації операції об’єднання функцій належності вихідної змінної.

max – максимум;

sum – сума;

probor – імовірнісне "АБО";

Custom… – власна реалізація операції об’єднання. Після натиснення необхідно в графічному вікні, що з’явилося, надрукувати ім’я функції, що реалізує цю операцію. Перед цим функцію треба створити як .m файл.

Меню Defuzzification

Це меню дозволяє вибрати метод дефазифікації. Для систем типу Мамдані запрограмовані наступні методи: **centroid** – центр ваги; **bisecto**r – медіана; **lom** – найбільший з максимумів; **som** – найменший з максимумів; **mom** – середнє з максимумів.

Користувач також має можливість установити власний метод дефазифікації. Для цього необхідно вибрати команду **Custom…** і в графічному вікні, що з'явилося, надрукувати ім'я функції, що реалізує цю операцію. Перед цим функцію треба створити як .m файл.

Membership Function Editor

Membership Function Editor (Редактор функцій належності) призначений для встановлення інформації про терми-множини вхідних і вихідних змінних, яка включає: кількість термів; найменування термів; тип і параметри функцій належності, у вигляді нечітких множин. Може бути викликаний з будь-якого GUI-модуля, використовуваного із системами нечіткого логічного виведення, командою **Membership Functions...** (Ctrl+2) меню **Edit**.

У FIS-редакторі відкрити редактор функцій належності можна також подвійним щигликом лівою кнопкою миші по області вхідної або вихідної змінних. Загальний вид редактора з указівкою функціонального призначення основних областей графічного вікна наведений на рис. 3.4.

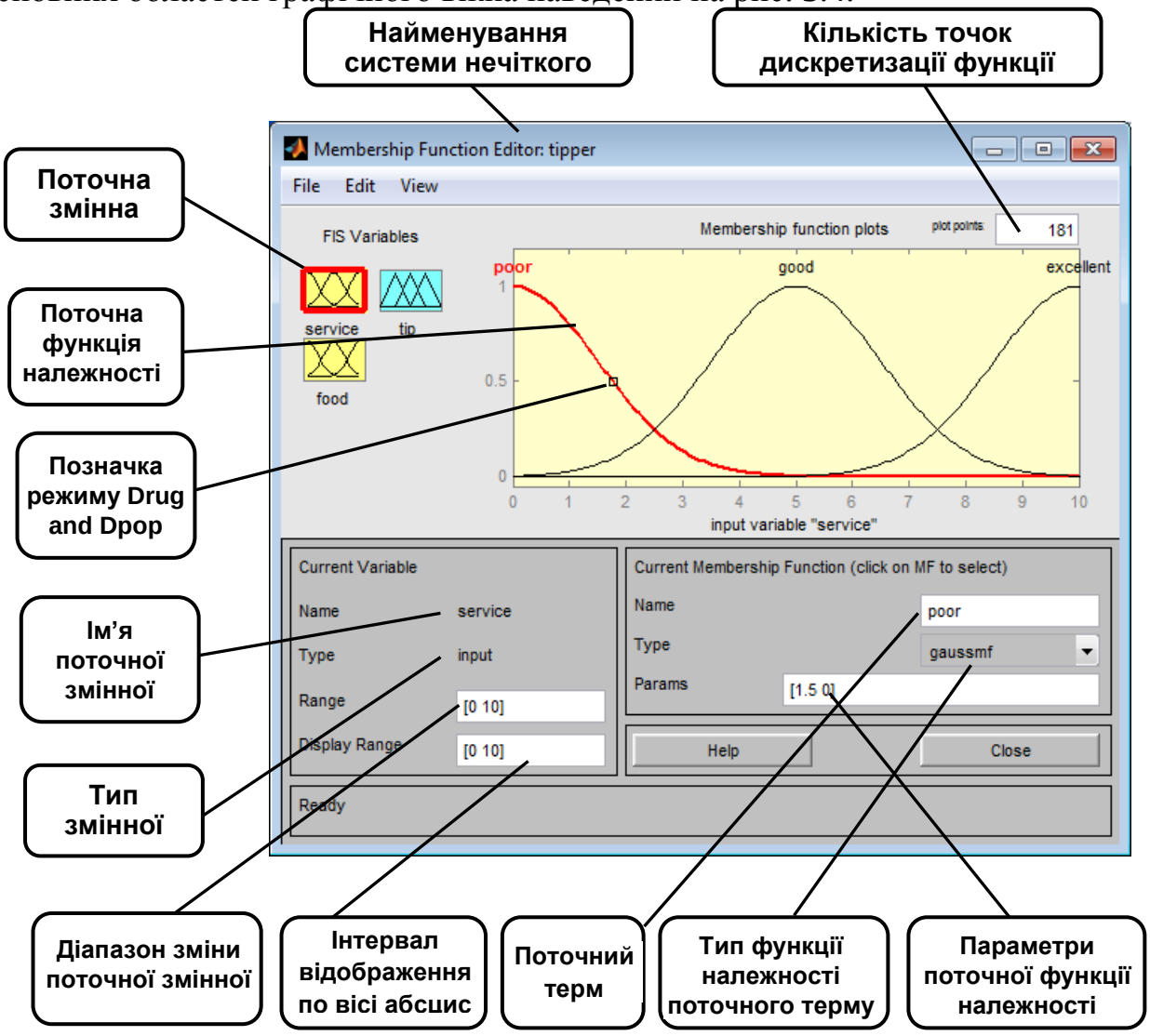


Рисунок 3.4 – Редактор функцій належності

У нижній частині графічного вікна розташовані кнопки Help і Close, що дозволяють викликати вікно довідки і закрити редактор, відповідно.

Редактор функцій належності містить чотири меню – **File, Edit, View, Type** і чотири вікна введення інформації – **Range, Display Range, Name і Params**. Ці чотири вікна призначені для завдання діапазону зміни поточної змінної, діапазону виведення функцій належності, найменування поточного лінгвістичного терму і параметрів його функції належності, відповідно.

Параметри функції належності можна підбирати й у графічному режимі, шляхом зміни форми функції належності за допомогою технології "Drug and drop". Для цього необхідно позиціонувати курсор миші на знаку режиму "Drug and drop", натиснути на ліву кнопку миші і не відпускаючи її змінювати форму функції належності. Параметри функції належності будуть перераховуватися автоматично.

Меню Edit

Undo (Ctrl+Z) – скасовує раніше зроблену дію.

Add MFs… – додавання термів в терм-множину поточної змінної для лінгвістичної оцінки поточної змінної. При виборі цієї команди з'явиться діалогове вікно, у якому необхідно вибрати тип функції належності і кількість термів. Значення параметрів функцій належності будуть встановлені автоматично таким чином, щоб рівномірно покрити область визначення змінної, заданої у вікні Range. При зміні області визначення у вікні Range параметри функцій належності будуть промаштабовані.

Add Custom MF… – додавання одного лінгвістичного терму, функція належності якого відрізняється від вбудованих. Після вибору цієї команди з'явиться графічне вікно, у якому необхідно надрукувати лінгвістичний терм (поле MF name), ім'я функції належності (поле M-File function name) і параметри функції належності (поле Parameter list).

Remove Selected MF – видаляє поточний терм із терм-множини поточної змінної. Ознакою поточної змінної є червона окантовка її прямокутника. Ознакою поточного терму є червоний колір його функції належності. Для вибору поточного терму необхідно провести позиціонування курсору миші на графіку функції належності і зробити щиглик лівою кнопкою миші.

Remove All MFs – видаляє всі терми з терм-множини поточної змінної.

FIS Properties… (Ctrl+1) – відкриває FIS-редактор.

Rules… (Ctrl+3) відкриває редактор бази правил (знань).

***Вікно* Range –** встановлення діапазону зміни лінгвістичної змінної.

***Вікно* Display Range –** встановлення діапазону відображення лінгвістичної змінної в робочій області вікна.

***Вікно* Name** присвоєння терму імені.

Меню Type

Вибір типу функції належності терму поточної змінної: trimf, trapmf, gbellmf, gaussmf, gauss2mf, sigmf, dsigmf, psigmf, pimf, smf, zmf.

***Вікно* Params –** встановлення параметрів функції належності терму поточної змінної. Наприклад значення трьох точок трикутної функції належності trimf.

Rule Editor (Редактор бази знань)

Rule Editor (*Редактор бази знань)* призначений для формування і модифікації нечітких правил. Редактор бази знань може бути викликаний з будь-якого GUI-модуля, використовуваного із системами нечіткого логічного виведення, командою **Rules…** меню **Edit** або натисканням клавіш Ctrl+3. У FIS-редакторі відкрити редактор бази знань можна також подвійним щигликом лівою кнопкою миші по прямокутнику з назвою системи нечіткого логічного виведення, розташованого в центрі графічного вікна.

Загальний вид редактора бази знань із указівкою функціонального призначення основних областей графічного вікна наведений на рис. 3.5.

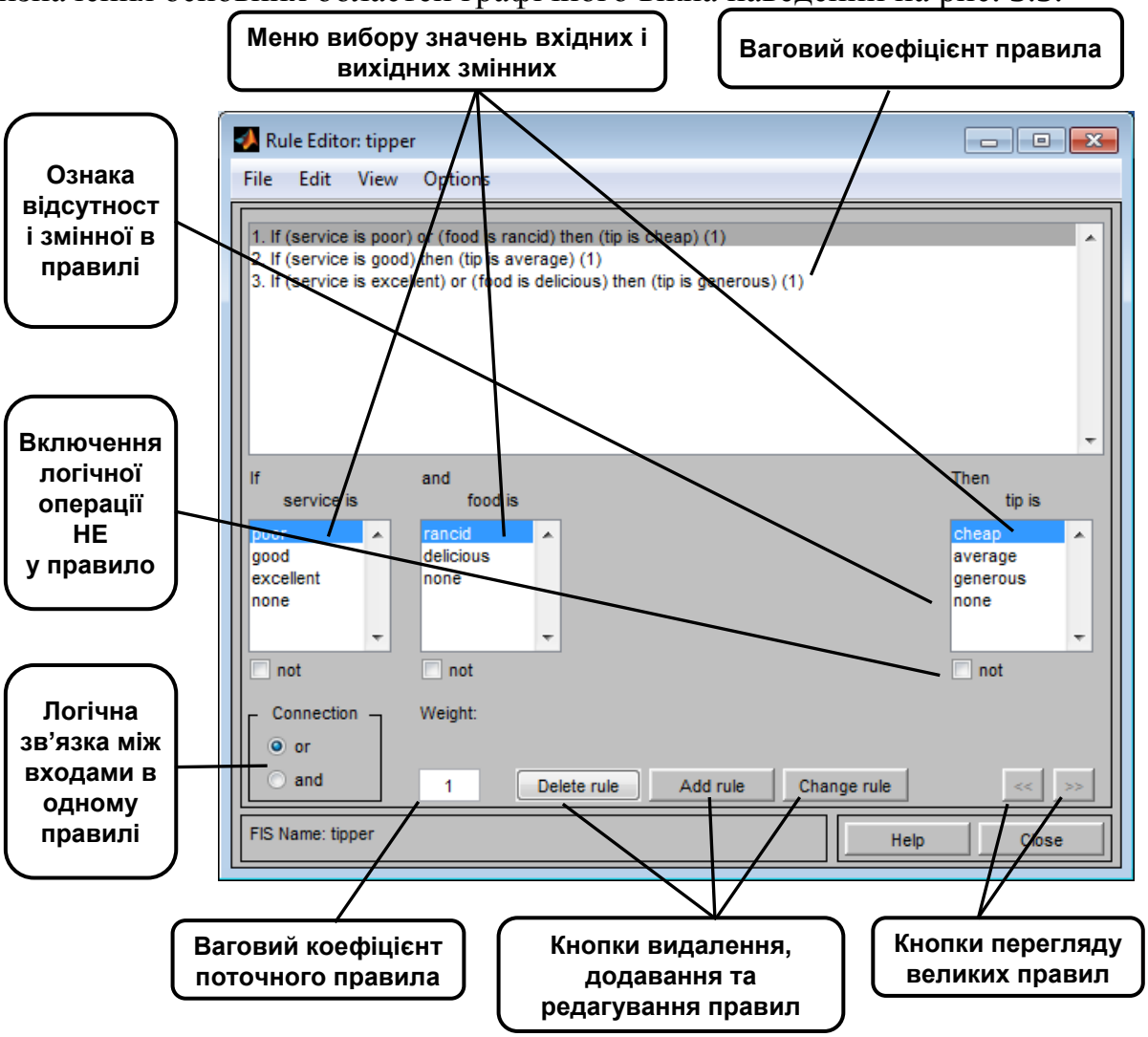


Рисунок 3.5 – Редактор бази правил (знань)

У нижній частині графічного вікна розташовані кнопки **Help** і **Close** для виклику вікна довідки і закриття редактора.

Редактор функцій належності містить чотири системних меню **File, Edit, View, Options,** меню вибору термів вхідних і вихідних змінних, поля установки логічних операцій **І, АБО, НЕ** і ваг правил, а також кнопки редагування і перегляду правил. Склад меню **File, Edit, View** наведений в попередніх розділах.

Меню Options

Language – встановлення мови текстових ідентифікаторів **English** (Англійська), **Deutsch** (Німецька), **Francais** (Французька).

Format – встановлення форматів правил бази знань: **Verbose** – лінгвістичний; **Symbolic** – логічний; **Indexed** – індексований.

Для введення нового правила в базу знань необхідно за допомогою миші вибрати відповідну комбінацію лінгвістичних термів вхідних і вихідних змінних, установити тип логічного зв'язування (**І** чи **АБО**) між змінними усередині правила, установити наявність чи відсутність логічної операції **НЕ** для кожної лінгвістичної змінний, увести значення вагового коефіцієнта правила і натиснути кнопку **Add Rule.** За замовчуванням установлені наступні параметри: логічне зв'язування змінних усередині правила – **І**; логічна операція **НЕ** – відсутня; значення вагового коефіцієнта правила – 1.

Можливі випадки, коли істинність правила не змінюється при довільному значенні деякої вхідної змінної, тобто ця змінна не впливає на результат нечіткого логічного виведення в даній області факторного простору. Тоді значення цієї змінної, як лінгвістичне, необхідно встановити **none**.

Для видалення правила з бази знань необхідно зробити однократний щиглик лівою кнопкою миші на цьому правилі та натиснути кнопку **Delete Rule**. Для модифікації правила необхідно зробити однократний щиглик лівою кнопкою миші на цьому правилі, потім установити необхідні параметри правила і натиснути кнопку **Edit Rule**.

Візуалізація нечіткого логічного виведення

Візуалізація нечіткого логічного виведення здійснюється за допомогою GUI-модуля **Rule Viewer** (рис. 3.6).

Цей модуль дозволяє проілюструвати хід логічного виведення за кожним правилом, одержання результуючої нечіткої множини і виконання процедури дефазифікації. **Rule Viewer** може бути викликаний з будь-якого GUI-модуля, використовуваного із системами нечіткого логічного виведення, командою **View rules …** (Ctrl+3) меню **View**.

Rule Viewer містить чотири меню – **File, Edit, View, Options,** дві області введення інформації – **Input і Plot points** та кнопки прокручування зображення вліво-вправо **left-right**, уверх-униз **up-down**. Склад меню **File, Edit, View** наведений в попередніх розділах. У нижній частині графічного вікна розташовані кнопки **Help** і **Close** для виклику вікна довідки і закриття редактора.

Кожне правило бази знань представляється у виді послідовності горизонтально розташованих прямокутників. При цьому перші два прямокутники відображають функції належності термів посилки правила (Якщо-частина правила), а останній третій прямокутник відповідає функції належності терму-наслідку вихідної змінної (То-частина правила).

Порожній прямокутник у візуалізації другого правила означає, що в цьому правилі посилання за змінною **food** відсутнє **food is none.** Жовте заливання графіків функцій належності вхідних змінних вказує наскільки значення входів,

відповідають термам даного правила. Для виведення правила у форматі Rule Editor необхідно зробити однократний щиглик лівої кнопки миші по номері відповідного правила. У цьому випадку зазначене правило буде виведено в нижній частині графічного вікна.

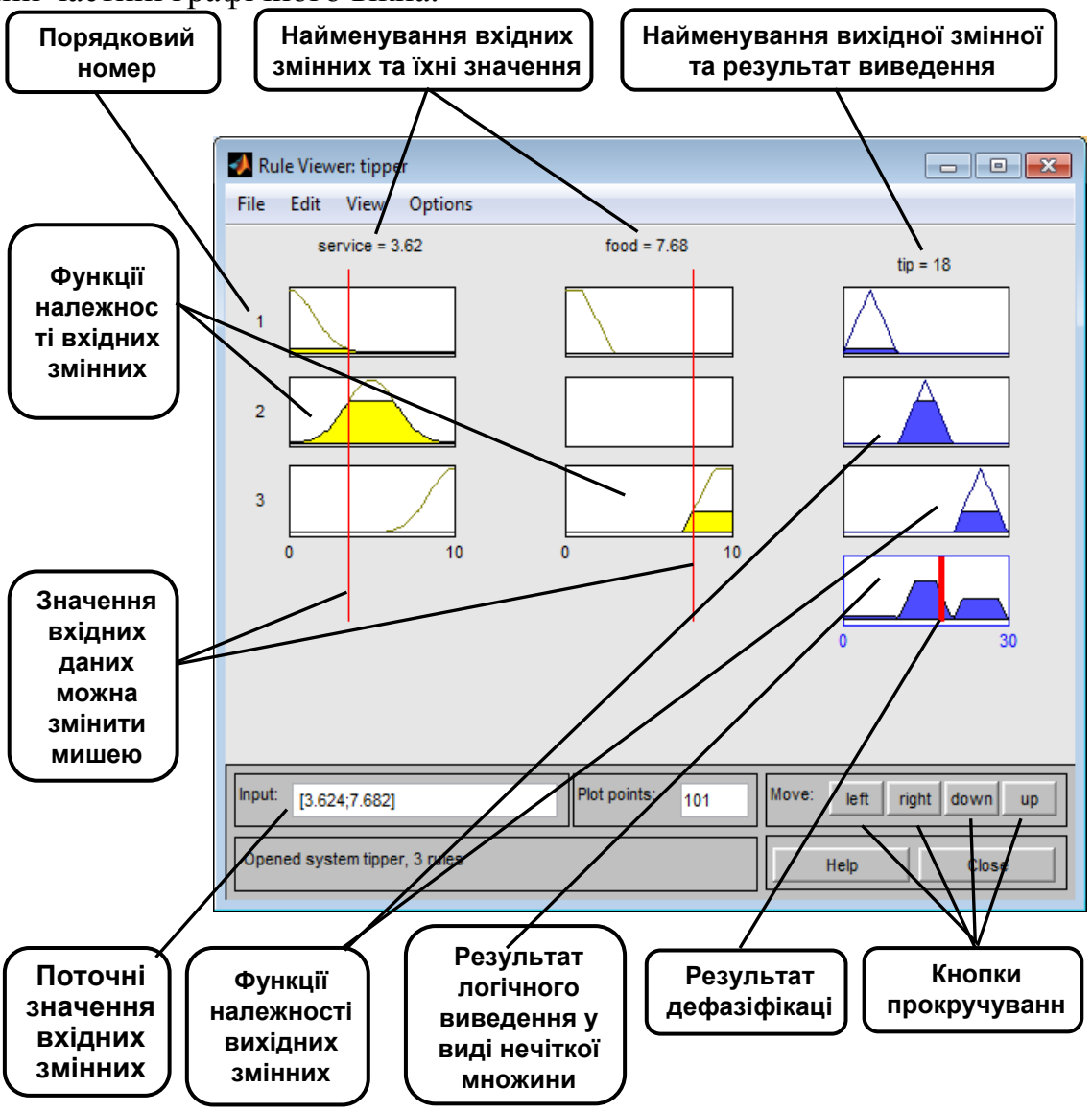


Рисунок 3.6 – Візуалізація логічного виведення за допомогою **Rule Viewer**

Блакитне заливання графіка функції належності вихідної змінної являє собою результат логічного виведення у вигляді нечіткої множини за даним правилом. Результуючу нечітку множину, що відповідає логічному виведенню за всіма правилами, показано в нижньому прямокутнику останнього стовпця графічного вікна. У цьому ж прямокутнику червона вертикальна лінія відповідає чіткому значенню логічного виведення, отриманого в результаті дефазіфікації.

Уведення значень вхідних змінних може здійснюватися двома способами: шляхом уведення чисельних значень в область **Input** за допомогою миші, шляхом переміщення ліній-покажчиків червоного кольору.

В останньому випадку необхідно позиціонувати курсор миші на червоній вертикальній лінії, натиснути на ліву кнопку миші і не відпускаючи неї перемістити покажчик на потрібну позицію. Нове чисельне значення відповідної

вхідної змінної буде перелічено автоматично і виведене у вікно **Input**. В області **Plot points** задається кількість точок дискретизації для побудови графіків функцій належності. Значення за замовчуванням – 101.

Практична робота 4
Основи програмування в MATLAB

Мета. Вивчити основи проблемно-орієнтованої системи програмування MATLAB, типи даних, матричні операції, способи створення функцій, придбати навички роботи з командно-графічним інтерфейсом системи.

Завдання

1. Згідно з варіантом (табл. 4.1) розробити файл сценарію обчислення функції з введенням даних користувачем і виведенням на екран.

2. Функцію реалізувати у виді файлу-функції з викликом з файлу сценарію.

Таблиця 4.1 – Варіанти завдання.

№ з/п	Функція
21.	Обчислення площі кола
22.	Обчислення площі прямокутника
23.	Обчислення периметру прямокутника
24.	Обчислення довжини кола
25.	Обчислення об’єму циліндра
26.	Обчислення об’єму конуса
27.	Обчислення площі сектора кола
28.	Обчислення об’єму шару
29.	Обчислення площі трапеції
30.	Обчислення об’єму паралелепіпеду
31.	Обчислити факторіал
32.	Пошук найбільшого елементу вектору
33.	Пошук найменшого елементу вектору
34.	Обчислення суми елементів
35.	Обчислення суми квадратів елементів
36.	Обчислення sin на основі функції Ейлера $\sin(x)=\frac{e^{ix}-e^{-ix}}{2i}$, i = $\sqrt{-1}$
37.	Обчислення cos на основі функції Ейлера $\cos(x)=\frac{e^{ix}+e^{-ix}}{2}$, i = $\sqrt{-1}$
38.	Обчислення tg на основі функції Ейлера $tg(x)=\frac{e^{ix}-e^{-ix}}{i(e^{ix}+e^{-ix})}$, i = $\sqrt{-1}$
39.	Обчислення ctg на основі функції Ейлера $ctg(x)=\frac{i(e^{ix}+e^{-ix})}{(e^{ix}-e^{-ix})}$, i = $\sqrt{-1}$
40.	Обчислення гіпотенузи прямокутного трикутника

Примітка. Номер варіанта співпадає з номером зі списку групи. Якщо номер зі списку перевищує кількість варіантів, варіант визначається як у попередніх роботах.

Зміст звіту

1. Титульна сторінка.
2. Тема практичного заняття.
3. Мета заняття.
4. Завдання.
5. Опис виконання завдань по пунктам з наданням скріншотів.
6. Тексти m-файлів
7. Висновки.

Контрольні питання

1. Для чого в MATLAB в кінці рядка використовується символ (;)?
2. Чим в MATLAB відрізняються команди (*) і (.*)?
3. У чому різниця між скалярним значенням, матрицею і вектором в MATLAB
4. Чим відрізняються файл-функція і файл-сценарій

Теоретичні відомості

4.1. Основні вікна MATLAB

На рис. 4.1 представлений зразок інтерфейсу з системою MATLAB.

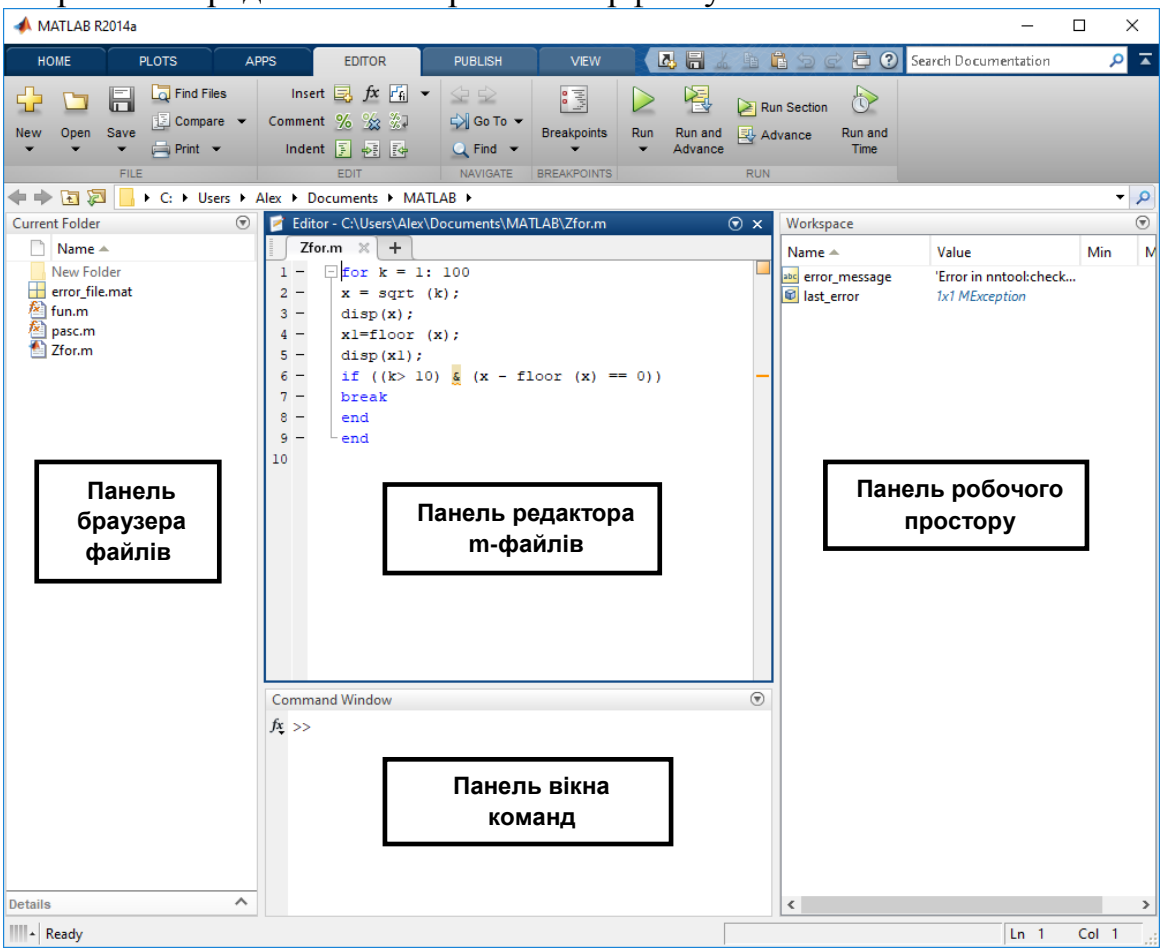


Рисунок 4.1 – Вікна MATLAB

Потужна інструментальна система MATLAB забезпечує процедурне, функціональне, логічне, структурне, об'єктно-орієнтоване і візуальне програмування. Вона базується на математико-орієнтованій мові надвисокого рівня, яка спрощує запис алгоритмів і відкриває нові методи їх створення.

Мова системи MATLAB за своєю структурою є командною. Команди виконуються в режимі інтерпретації.

Вікно команд

MATLAB створений таким чином, що будь-які (іноді дуже складні) обчислення можна виконувати в режимі прямих обчислень, тобто без підготовки програми користувачем. При цьому MATLAB виконує функції суперкалькулятора і працює в режимі командного рядка.

Робота з системою носить діалоговий характер і відбувається за правилом «задав питання – отримав відповідь». Користувач набирає на клавіатурі вираз, що обчислюється, редагує його (якщо потрібно) в командному рядку і після натисканням клавіші ENTER у командному вікні отримує результат (ans =) з числовими даними. Якщо в кінці командного рядка стоїть “ **;** ”? то результат не виводиться.

Вікно робочого простору

У вікні робочого простору відображаються змінні і їх значення на поточний момент часу роботи з командним вікном. Значення елементів матриці можна побачити в окремому вікні, натиснувши на ім’я матриці.

Браузер файлової структури

Для перегляду файлової структури MATLAB служить спеціальний браузер файлової системи (Path Browser), який запускається при звичайному завантаженні системи. Якщо був встановлений спрощений інтерфейс, то для запуску браузера файлової системи використовується вікно Current Folder. На рис. 4.1 в лівій частині показано вікно цього браузера.

Вікно редактора m-файлів

За допомогою мови системи MATLAB можна створювати текстові модулі-функції та модулі-сценарії (програми). Файли, де зберігаються такі модулі, мають розширення ***.m** і називаються М-файлами, а функції, що в них знаходяться – М-функціями. У системі є величезна бібліотека М-функцій в текстовому форматі, які можна модифікувати для досягнення бажаних цілей. Користувач може створювати власні М-функції і включати їх в систему.

При вирішенні серйозних завдань виникає необхідність збереження використовуваних послідовностей обчислень, а також їх подальших модифікацій. Для вирішення цього завдання система MATLAB має у своєму складі потужну мову програмування високого рівня.

Для редагування файлів програм може використовуватися будь-який текстовий редактор, а також спеціальний багатовіконний редактор MATLAB. Редактор програм системи має наступні можливості:

- колірне підсвічування синтаксису, що дозволяє виявити помилки;
- синтаксичний контроль на стадії підготовки М-файлу;
- установка точок переривання при інтерпретації команд;
- автоматична нумерація рядків програми для видачі повідомлень.

4.2. Основи редагування та налагодження m-файлів

Інтерфейс редактора m-файлів

Для підготовки, редагування та налагодження m-файлів служить спеціальний багатовіконний редактор. Він виконаний як типовий додаток Windows. Редактор можна викликати командою edit з командного рядка або з меню New → Script. Після цього у вікні редактора можна створювати свій файл, користуватися засобами його налагодження і запуску.

На рис. 4.1 показано вікно редактора з текстом простого файлу у вікні редагування та налагодження програмного коду.

Призначення кнопок панелі інструментів редактора:

- New – створення нового m-файлу;
- Open – завантаження нового файлу;
- Save – збереження файлу на диску;
- Print – друк вмісту поточного вікна редактора;
- Breakpoints – установка / скидання точок переривання;
- Run – запуск програми на виконання і збереження;

Підготовлений текст файлу в редакторі MATLAB треба записати на диск. Для цього використовується меню Save. Після запису файлу на диск, команда Run в меню редактора стає активною (до запису файлу на диск вона пасивна) і дозволяє провести запуск файлу. Запустивши команду Run, можна спостерігати виконання m-файлу.

Для зручності роботи з редактором рядки програми в ньому нумеруються в послідовному порядку. Редактор є багато віконним. Вікно кожної програми оформляється як вкладка.

Кольорові виділення і синтаксичний контроль.

Редактор m-файлів виконує синтаксичний контроль програмного коду під час введення тексту. При цьому використовуються наступні колірні виділення:

- ключові слова мови програмування – синій колір;
- оператори, константи і змінні – чорний колір;
- коментарі після символу % – зелений колір;
- символьні змінні (в апострофах) – зелений колір;
- синтаксичні помилки – червоний колір.

Завдяки кольоровим виділенням ймовірність синтаксичних помилок знижується.

Робота з точками переривання

Основним прийомом налагодження m-файлів є установка в їх тексті точок переривання (Breakpoints), коли програма виконується до цієї точки і зупиняється. В такому режимі ми можемо подивитись на результати, що досягнуті на цей момент, наприклад, поточні значення змінних і зробити висновок про правильність чи не правильність роботи програми, поті запустити далі до нової точки переривання і так до завершення програми. Точки переривання встановлюються (і скидаються) в меню Breakpoints за допомогою кнопок Set/Clear. Скидання всіх точок переривання забезпечується кнопкою Clear All.

Математичні обчислення

MATLAB – це математичний пакет прикладних програм, заснований на використанні матриць. Пакет містить велику бібліотеку програм з чисельних методів, використовує дво- і тривимірну графіку, а також формати мов високого рівня. Основним об'єктом MATLAB є матриця, навіть якщо це проста змінна, і всі операції здійснюються над матрицями.

Арифметичні операції

\+ Додавання
– Віднімання
* Добуток
/ Ділення
^ Піднесення до степеня

Постійні: **pi**, **e**, **i** (pi = 3.14159…, e = 2,71828…., i = $\sqrt{-1}$)

Приклад.

```
>> (2 + 3 * pi) / 2*i
ans = 0.0000 + 5.7124i
```

Однакові за позначенням операції над матрицями і простими змінними за результатом не співпадають, тому, що результат в загальному випадку теж матриця. Щоб оперувати зі звичними змінними перед знаком операції потрібно ставити крапку.

Вбудовані функції

Основні математичні функції, наявні в MATLAB: abs (#) cos (#) exp (#) log (#) log10 (#) cosh (#) sin (#) tan (#) sqrt (#) floor (#) acos (#) tanh (#).

Наступний приклад ілюструє, як можна комбінувати функції і арифметичні операції.

Приклад. >> 3 * cos (sqrt (4.7))

ans = –1.6869

Опис інших функцій можна знайти в діалоговому режимі, набравши в командному вікні команду help і ім'я функції.

Приклад. >> help sin

За замовчуванням результат обчислення функції подається приблизно п'ятьма десятковими значущими цифрами. Команда format long дозволяє вивести приблизно 15 десяткових значущих цифр.

Приклад. >> format long

>> 3 * cos (sqrt (4.7))

ans = –1.68686892236893

Оператори присвоювання

Для присвоювання змінним результатів різних дій застосовується знак рівності.

Приклад. > a = 3–floor (exp (2.9))

a = –15

Приклад. >> b = sin (a);

>> 2 * b ^ 2

ans =

0.8457

Оператори відносин

== Дорівнює
~= Не дорівнює
< Менше
> Більше
<= Менше або дорівнює
>= Більше або дорівнює

Логічні оператори

~ Not (доповнення)
& And (істина, якщо істинні обидва операнди)
| Or ((істина, якщо один з двох або обидва операнди істинні)

Булеві величини

1 True
0 False

Матриці

Всі змінні в MATLAB інтерпретуються як матриці або масиви. Матриці можна вводити безпосередньо:

```
Приклад.   >> A = [1 2 3; 4 5 6; 7 8 9]
             A = 1 2 3
             4 5 6
             7 8 9
```

Крапка з комою використовується для поділу рядків матриці. Елементи матриці слід розділяти одиночним пропуском (пробілом).

Маніпулювання з елементами матриці.

```
Приклад.  >> A (2,3)  % Виділення одного елемента матриці A
          ans =
          6
Приклад.  >> A (1: 2, 2: 3) % Виділення підматриці A
          ans =
          2 3
          5 6
Приклад.  % Присвоєння нового значення елементу матриці A
          >> A (2,2) = tan (7.8);
```

Матриці можна формувати, використовуючи вбудовані функції.

```
Приклад.  >> Z = zeros (3,5); % Створення нульової матриці розміру 3x5.
          >> X = ones (3,5); % Створення матриці розміру 3x5, що
                             % складається з одиниць.
          >> Y = 0: 0.5: 2   % Вивід на екран матриці розміру 1x5.
          Y =
          0 0.5000 1.0000 1.5000 2.0000
          >> cos (Y) % Створення матриці розміру 1x5, кожен елемент
                     % якої є cos від Y
          ans =
          1.0000 0.8776 0.5403 0.0707 -0.4161
```

Додаткові команди для матриць можна знайти, використовуючи довідку в діалоговому режимі або документацію до пакету прикладних програм.

Операції з матрицями

+ Додавання
– Віднімання
* Добуток
^ Піднесення до степеня
' Спряження та транспонування (стовпці замінюються на рядки)

Приклад.
```
>> B = [1 2, 3 4];
>> C = B'              % C дорівнює транспонованій матриці B
C =
1 3
2 4
% Приклад операцій над матрицями
% добуток попередніх матриць піднести до кубу і помножити на 3
>> 3 * (B * C) ^ 3
ans =
13080  29568
29568  66840
```

Операції з масивами

Однією з найбільш корисних властивостей пакета MATLAB є можливість виконувати операції над окремими елементами матриці. Така операція була продемонстрована вище, коли елементи матриці розміру 1x5 записувалися як cos від елементів іншої матриці. Операції додавання, віднімання і добутку матриці на скаляр завжди здійснюються поелементно, але це не відноситься до операцій множення, ділення і піднесення матриці до ступеня. Дані три операції можна проводити поелементно, як і попередні, але з крапкою перед символом операції: **.*, ./ .^.** Важливо зрозуміти, як і коли саме їх можна використовувати. Операції над масивами є вирішальними для ефективної побудови та виконання програм пакетом MATLAB.

Приклад.
```
>> A = [1 2, 3 4];
>> A ^2 % Добуток матриць A*A або піднесення в квадрат
ans =
7 10 15 22
>> A .^2    % Квадрат кожного елемента A
ans =
1 4
9 16
>> cos (A./2)  % Ділення кожного елемента A на 2 і
                % обчислення cos від кожного елемента
ans =
0.8776 0.5403 0.0707 -0.4161
```

Введення значень з клавіатури і виведення на екран

Для організації діалогового вводу/виводу використовуються функції input (введення з клавіатури) і disp (виведення значення на екран).

Функція disp призначена для виведення її параметра на екран.

Функція disp має наступний синтаксис:

disp (Значення)

Приклад. >> disp('Hello') % Виведення рядка символів

Функція input має наступний синтаксис:

Ім’я змінної = input ('рядок')

При виконанні цієї функції спочатку виводиться рядок, потім відбувається зупинка роботи програми і очікується введення значення. Введення підтверджується натисканням клавіші Enter, після чого введене значення присвоюється змінній.

Приклад. % Введення і обчислення квадрату числа

```
>> x=('Input number: ')
z=input(x);
y=z.^2;
disp(y)
```

Цикли і умовні оператори

Крім програм з лінійною структурою, інструкції яких виконуються строго по порядку, MATLAB дозволяє створювати програми, структура яких нелінійна. Для створення таких програм застосовуються наступні умовні оператори: **if, for, while**.

Синтаксис оператора **for**:

for (**var** = вираз)
Послідовність операторів
end

Цикли типу **for ... end** зазвичай використовуються для організації обчислень із заданим числом повторюваних циклів.

Вираз найчастіше записується у вигляді **s: d: e**, де: **var** – ім’я змінної циклу, **s** – початкове значення змінної циклу, **d** – прирощення цієї змінної та **e** – кінцеве значення, при досягненні якого цикл завершується. За замовчуванням **d** дорівнює 1.

Приклад. % Виведення на екран значень **g** з діапазону –1…0, з кроком – 0.1

```
>> for g = 1.0: –.1: 0.0
       disp(g)
   end
```

Синтаксис оператора **while**:

while Умова
Послідовність операторів
end

Цикл типу while... end виконується до тих пір, поки залишається істинним умова:

Приклад. % Обчислення суми чисел віл 1 до 100

```
n = 1;
nSum = 1;
while n < 100
  n = n + 1;
```

```
    nSum = nSum + n;
end
```

Синтаксис оператора **if**:

if Умова
    Послідовність операторів
else
    Послідовність операторів
end

Приклад.

```
x=('Input number: ');
z=input(x);
if z>5
   F=z+2;
else
  F=z^2;
end;
disp(F);
```

Для зупинки програми використовується оператор pause. Він використовується в таких формах:

1) pause – зупиняє обчислення до натискання будь-якої клавіші;
2) pause (N) – зупиняє обчислення на N секунд;
3) pause on – включає режим обробки пауз;
4) pause off – вимикає режим обробки пауз.

М-файли сценаріїв і функцій

М-файли системи MATLAB діляться на два класи:

- файли-сценарії (програми), що не мають вхідних параметрів;
- файли-функції, що мають вхідні параметри.

Файл-сценарій або Script-файл не має списку вхідних параметрів. Він використовує глобальні змінні, тобто такі змінні, значення яких можуть бути змінені в будь-який момент сеансу роботи і в будь-якому місці програми. Для запуску файлу-сценарію з командного рядка MATLAB достатньо вказати його ім’я в цьому рядку.

Файл-сценарій має наступну структуру:

% Основний коментар – один рядок (обов’язковий)

% Додатковий коментар – будь-яке число рядків (не обов’язковий)

Тіло файлу – будь-які вирази, команди і оператори.

Файл-функція відрізняється від файлу-сценарію насамперед тим, що створена ним функція має вхідні параметри, список яких вказується у круглих дужках. Використовувані в файлах-функціях змінні і імена параметрів є локальними змінними, зміна значень яких в тілі функції не впливає на значення змінних з таким ж іменами за межами функції.

Файл-функція має наступну структуру:

function var = ***ім’я т-файлу*** **(Список параметрів переданих значень)**

% Основний коментар – один рядок (обов’язковий)

% Додатковий коментар – будь-яке число рядків (необов'язковий)

Тіло файлу – будь-які вирази, команди і оператори.

var = вираз.

Остання інструкція тіла "var = вираз" вводиться, якщо потрібно, щоб функція повертала результат обчислень. Якщо необхідна більша кількість вихідних параметрів, структура модуля буде мати наступний вигляд:

function [var1, var2, ...] = *ім'я m-файлу* (Список параметрів переданих значень)

% Основний коментар – один рядок (обов'язковий)

% Додатковий коментар – будь-яке число рядків (необов'язковий)

Тіло файлу – будь-які вирази, команди і оператори.

var1 = вираз

var2 = вираз

Імена **var, varl, var2**, ... для значень, що повертаються є глобальними або відомими в тілі програми, яка викликає М-функції.

Приклад. Визначимо функцію fun (x) = 1 + x – x^2 / 4 в М-файлі fun.m.

В редакторі вона записується в такий спосіб.

```
function y = fun(x)
% Example M-script
%   Calculate y=1+x–x^2/4
y=1+x–x.^2./4;
end
```

Коментар з М-файлу можна вивести на екран за допомогою команди **help *ім'я m-файлу***, причому вивід проводиться до першого порожнього рядка.

Для позначення змінних можна вживати різні літери і для назви функцій – різні імена, але слід використовувати один і той же формат. Функцію, одного разу записану як М-файл з ім'ям **fun.m**, можна викликати в MATLAB Command Window так само, як будь-яку іншу функцію.

```
Приклад.   >> p=1
           >> fun(p)
           ans=
               1.7500
Приклад.   >> cos (fun (3))
ans =
  -0.1782
```

Корисним і ефективним способом обчислення функцій є використання команди feval. Вона вимагає, щоб функція викликалася, як рядок.

```
Приклад.   >> feval ('fun', 4)
           ans =
           1
```

Практична робота 5
Дослідження багатошарового персептрона для моделювання логічних функцій

Мета. Отримати практичні навички створення і навчання багатошарового персептрона в пакеті MatLab.

Завдання

Створити й навчити нейронну мережу, здатну розв’язувати логічне завдання. Перевірити працездатність нейронної мережі.

Для розв'язку завдань, представлених у даному занятті, рекомендується використовувати нейронну мережу типу багатошаровий персептрон, що складається з трьох шарів: вхідний шар з 4-ма нейронами, прихований шар з двома нейронами і вихідний шар з одним нейроном (4-2-1). Порядок виконання роботи.

1. Скласти таблицю істинності для заданого логічного виразу (X_1, X_2, X_3, X_4, приймають значення 0 або 1).
2. Ввести в якості вхідних значень всі можливі сполучення X_1, X_2, X_3, X_4, а в якості вихідних значень – значення виходу в таблиці істинності.
3. Створити нейронну мережу з використанням операторів MATLAB.
4. Провести навчання нейронної мережі і вивести значення вагових коефіцієнтів.
5. Протестувати отриману нейронну мережу, задаючи в якості входу три варіанти значень X_1, X_2, X_3, X_4.
6. За результатами моделювання створити **.m** – файл з програмою.
7. Проаналізувати роботу нейронної мережі.

Таблиця 5.1 – Варіанти завдань

№ варіанту	Логічна функція
1	$(X_1 \vee X_2) \wedge X_3 \wedge X_4$
2	$X_1 \wedge X_2 \wedge (X_3 \vee X_4)$
3	$X_1 \wedge (X_2 \vee X_3) \vee X_4$
4	$X_1 \vee (X_2 \vee X_3) \wedge X_4$
5	$(X_1 \wedge X_2) \vee (X_3 \wedge X_4)$
6	$(X_1 \vee X_2) \wedge X_3 \vee X_4$
7	$X_1 \vee (X_2 \vee X_3) \wedge X_4$
8	$X_1 \wedge X_2 \vee X_3 \wedge X_4$
9	$X_1 \vee X_2 \wedge X_3 \vee X_4$

Для обчислення логічної функції можна скористатись таблицями 2 і 3.

Зміст звіту

1. Титульна сторінка.
2. Тема практичного заняття.
3. Мета заняття.
4. Номер варіанту та завдання до роботи.

5. Опис виконання завдань по пунктам з наданням рисунків і скріншотів.
6. Текст створеного *.m файлу.
7. Висновки, що відображують результати та їх критичний аналіз.

Контрольні запитання

1. Модель штучного нейрона.
2. Основні типи активаційних функцій.
3. Спосіб обчислення коефіцієнтів w_{ij} в алгоритмах навчання.
4. Постановка задачі навчання ШНМ.
5. Параметри алгоритму навчання.

Таблиця 5.2 – Істинність операції І (∧)

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Таблиця 5.3 – Істинність операції АБО (∨)

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	1

Теоретичні відомості

Штучна нейронна мережа (ШНМ) – набір елементарних нейроподібних перетворювачів інформації (нейронів), з'єднаних між собою.

На вхід формального нейрона надходить набір вхідних сигналів $x_1, x_2, ..., x_n$ або вхідний вектор x. Кожен вхідний сигнал помножується на відповідну вагу $w_1, w_2,..., w_n$. Вага зв'язку є скалярною величиною, додатною для збудливих і від'ємною для гальмуючих зв'язків. Зважені вагами зв'язків вхідні сигнали надходять на блок сумації, де здійснюється їх алгебраїчне підсумовування та визначається рівень збудження нейроподібного елемента y:

$$y = f\left(\sum_{i=o}^{n} W_i x_i + \theta\right) \qquad (5.1)$$

де f – нелінійна функція активації, θ – деякий постійний зсув.

Для більшості моделей ШНМ такий нейрон є основним конструкційним елементом і часто представляється графічно рис. 1.

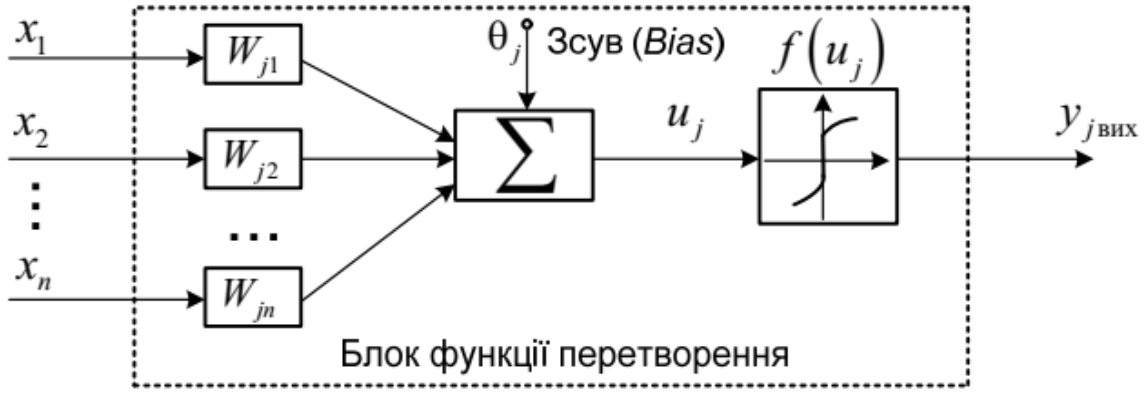


Рисунок 5.1 – Модель штучного нейрона (одношаровий персептрон)

Одношаровий персептрон є одним з найпростіших варіантів ШНМ і містить лише один нейрон.

В якості функції f за звичай використовуються найпростіші нелінійні функції:

бінарна (порогова): $$f(\mathrm{x}) = \begin{cases} 1, \text{ при } \mathrm{x} > 0, \\ 0, \text{ при } \mathrm{x} < 0, \end{cases} \tag{5.2}$$

або сигмоїдна: $$f(x) = \frac{1}{1+e^{-\alpha x}}. \tag{5.3}$$

Одним з типів нейронних мереж є мережі прямого поширення або багатошаровий персептрон рис. 2.

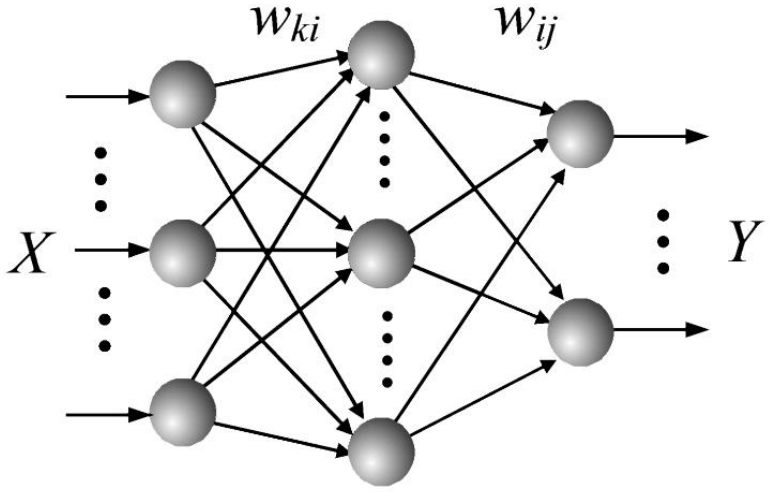


Рисунок 5.2 – Структура мережі прямого поширення
(багатошаровий персептрон)

Штучна нейронна мережа прямого поширення складається з вхідного шару, декількох схованих шарів і вихідного шару і описується формулою:

$$y_j = f\left(\sum_{i=o}^{n} W_{ji} x_i + \theta_j\right) \tag{5.4}$$

Навчання нейронної мережі.

ШНМ прямого поширення навчаються за методом навчання з учителем, коли для відомих даних на вході мережі ми знаємо, який результат повинен бути на виході. В такому випадку навчити ШНМ означає знайти такі вагові коефіцієнти w_{ji} і зсуви θ_j для кожного нейрону мережі з формули (4), щоб різниця між бажаними і обчисленими результатами на виході не перевищувала заданий наперед рівень помилки. В алгоритмах навчання ця помилка описується функцією втрат (loss function). Обчислення w_{ji} і θ_j здійснюється по крокам за формулами:

$$w_{ij}(n+1) = w_{ij}(n) - \beta \Delta w_{ij} \tag{5.5}$$

$$\theta_j(n+1) = \theta_j(n) - \beta \Delta \theta_j \tag{5.6}$$

На першому кроці ці значення обираються випадковим чином, тому повторне навчання на тих же даних буде давати різні результати. Спосіб обчислення значень Δw_{ji} і $\Delta \theta_j$ визначається алгоритмом навчання. Суттєве значення в цих формулах має коефіцієнт β, що зветься коефіцієнтом швидкості навчання. Його встановлюють у межах $0 < \beta < 1$ за звичай $\beta = 0.1$. При великих значеннях β є ймовірність проскочити оптимальне значення ваг, при малих значеннях β навчання буде відбуватись довго. Досвід роботи з ШНМ показав, що не для любих даних мережу можна навчити, тому, щоб уникнути нескінченого циклу вводять обмеження на кількість епох. Епоха – це коли в алгоритмі навчання на вхід ШНМ поступають усі дані для навчання. Крім цього існує явище перенавчання, коли кажуть, що мережа не навчилась, а просто запам’ятала вхідні дані. Для виявлення цього явища навчальну вибірку розбивають на основну або тренувальну (*Train*), перевірочну (*Validation*) для виявлення перенавчання і тестову (*Test*) для оцінки якості навченої мережі. Перенавчання фіксується на кроці, коли функція втрат продовжує зменшуватись для тренувальної вибірки і починає збільшуватись для перевірочної. Крім цього для припинення навчання можна задавати максимальний час навчання і специфічні параметри алгоритмів навчання, наприклад, мінімальний рівень градієнту функції втрат, що використовується при обчисленні Δw_{ji} і $\Delta \theta_j$.

Створення нейронної мережі.

Вибір структури нейронної мережі являє собою окреме завдання й полягає у виборі топології мережі (кількості шарів і кількості нейронів в кожному шарі) й функцій активації кожного нейрона. За звичай функція активації однакова для всіх нейронів шару.

Створення навчальної вибірки. Навчальна вибірка повинна мати достатній розмір, щоб забезпечити ефективність навчання, враховуючи, що в алгоритмах навчання вона випадковим чином розбивається на тренувальну, перевірочну і тестову. Зазвичай співвідношення між Train, Validation та Test вибірками: 70: 20: 10.

Функція втрат (*loss function*) показує якість навчання нейронної мережі або ступінь відповідності множини виходів ШНМ бажаній множині виходів.

Епоха – крок навчання ШНМ, за який на вхід послідовно представляються усі вектори навчальної вибірки.

Приклад створення і навчання нейронної мережі

```
% Навчальна вибірка
x=[0 0 0 0 1 1 1 1;
   0 0 1 1 0 0 1 1;
   0 1 0 1 0 1 0 1];
y=[0 0 0 1 0 0 0 1];
% Структура і параметри нейронної мережі прямого поширення (newff)
% кількість шарів – 2;
% кількість нейронів у вхідному шарі – 3;
% кількість нейронів у схованому шарі – 2;
% функція активації нейронів вхідного і схованого шару – сигмоїда;
% мережа буде навчатися за алгоритмом Левенберга-Макварта
net=newff(x,y,[3,2],{'logsig','logsig'},'trainlm');
net.trainparam. show=25; % кількість епох відображення на графіку
% значення функції втрат під час навчання
net.trainparam.lr= 0.1;          % коефіцієнт швидкості навчання
net.trainparam. epochs=500;      % максимальна кількість епох навчання
net.trainparam. goal=0.0001;     % значення похибки функції втрат

                % параметри розподілу навчальної вибірки
net.divideParam.trainRatio = 100/100; % тренувальна вибірка
net.divideParam.valRatio = 0/100;          % валідаційна вибірка
net.divideParam.testRatio = 0/100;         % тестова вибірка
net=train(net, x, y); % навчання мережі
w1=net.iw{1,1}; % вивід вагових коефіцієнтів і зсувів нейронів вхідного шару
display(w1);
b1=net.b{1,1};
display(b1);
w2=net.lw{2,1}; % вивід вагових коефіцієнтів і зсувів нейронів другого шару
display(w2);
b2=net.b{2,1};
display(b2);
a=sim(net, x); % обчислення помилки між бажаним значенням і обчисленим
display(a);
display(y);
er=a-y;
plot(er);
```

Після початку навчання з’являється вікно Neural Network Training (nntraintool) (рис. 4.), в якому відображається структура створеної нейронної мережі (панель Neural Network), деякі параметри алгоритму навчання (панель Algorithms), результуючі значення досягнутих параметрів навчання (панель Progress), результати навчання у виді графіків (панель Plots), Якщо навчання відбувається довго, то його можна призупинити, натиснувши кнопку Stop Training.

Процес зміни функції втрат під час навчання і результат можна спостерігати у вікні Neural Network Training Performance рис. 3, яке викликається з вікна Neural Network Training (nntraintool) натисканням кнопки Performance рис. 4.

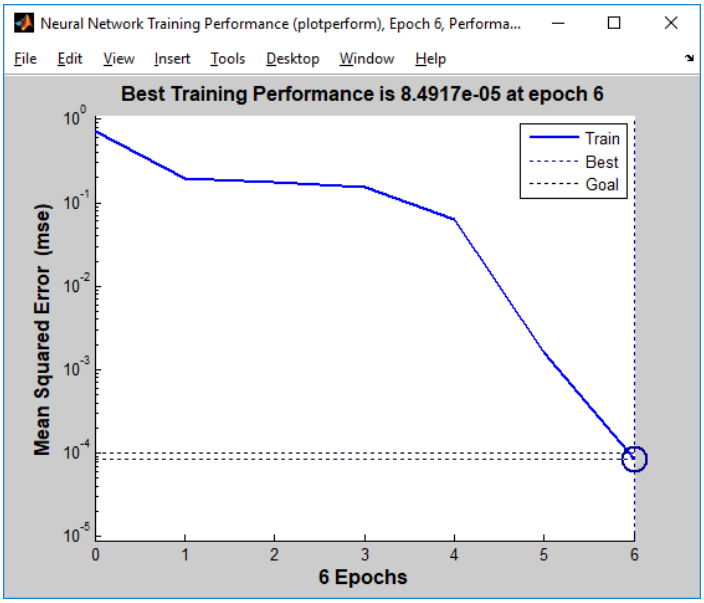


Рисунок 5.3 – Вікно зміни функції втрат при навчання нейронної мережі

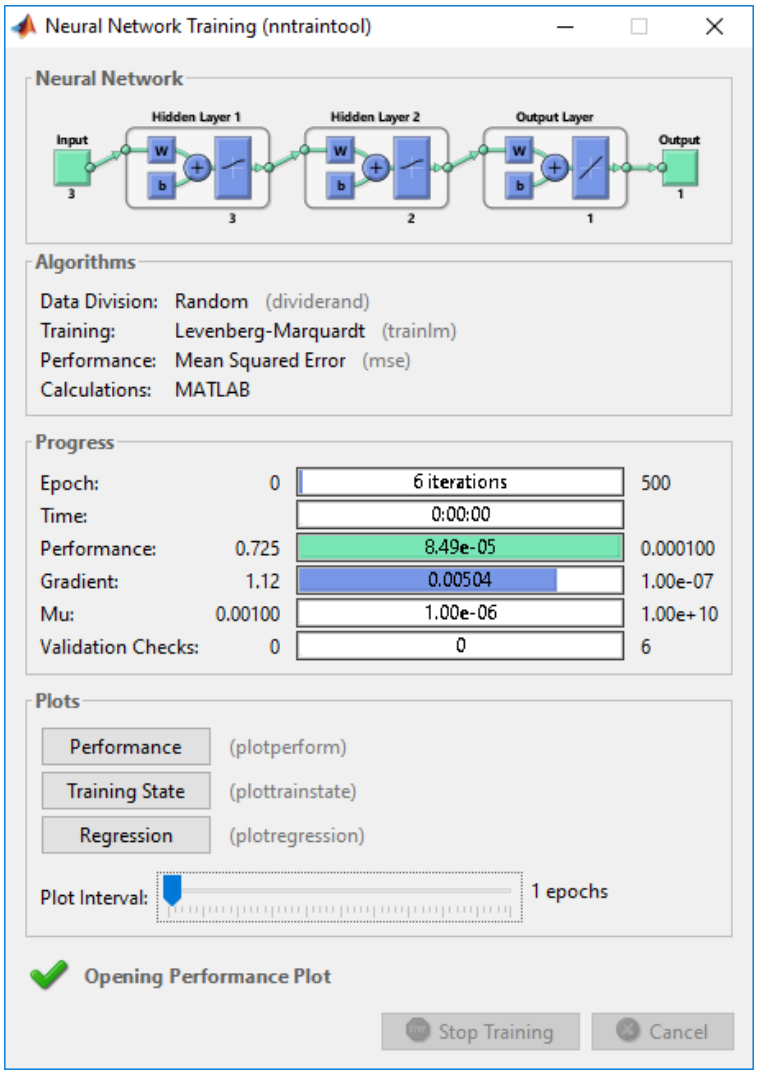


Рисунок 5.4 – Вікно контролю навчання нейронної мережі

На рис. 5.3 і рис. 5.4 представлений результат навчання мережі з наведеного вище прикладу. Функція newff примусово добавляє на виході один нейрон з лінійною функцією активації.

Таблиця 5.3 – Опис структури нейромережі у MATLAB

Поле структури	**Опис значення поля**
numInputs	кількість входів мережі
numLayers	кількість (схованих) шарів мережі без урахування вхідного шару
biasConnect, inputConnect, layerConnect, outputConnect, targetConnect	булеві масиви, що визначають зв’язки між структурними елементами мережі
numOutputs	кількість виходів мережі
numTargets	кількість цільових ознак
numInputDelays	кількість затримок вхідного шару
numLayerDelays	кількість затримок схованих шарів
inputs	входи мережі
layers	шари мережі
outputs	виходи мережі
targets	цільові ознаки

biases	масив порогів нейронів мережі
inputWeights	масив вагових коефіцієнтів вхідного шару мережі
layerWeights	масив вагових коефіцієнтів схованих шарів мережі
adaptFcn	ім’я функції адаптації нейронів
initFcn	ім’я функції ініціалізації мережі
performFcn	ім’я цільової функції навчання (помилки)
trainFcn	ім’я функції, що реалізує процес навчання
adaptParam	параметри адаптації мережі
initParam	параметри ініціалізації мережі
performParam	параметри цільової функції мережі
trainParam.epochs	максимально допустима кількість ітерацій навчання (епох)
trainParam.goal	максимально допустиме значення цільової функції навчання
trainParam.max_fail	максимально допустима кількість відмов у процесі навчання мережі
trainParam.mem_reduc	коефіцієнт, що регулює (зменшує) використання пам’яті при навчанні нейромереж
trainParam.min_grad	мінімально допустиме значення градієнту цільової функції
trainParam.mu, trainParam.mu_dec, trainParam.mu_inc, trainParam.mu_max	параметри методу Левенберга-Марквардта
trainParam.show	кількість ітерацій, через яку будуть відображатися зміни стану процесу навчання мережі
trainParam.time	максимально допустимий час навчання мережі у секундах
iw	масив значень ваг вхідного (першого) шару
lw	масив значень ваг схованих шарів
b	масив значень порогів нейронів
userdata	дані користувача
divideParam.trainRatio	розмір навчальної вибірки відносно всіх початкових даних (приклад: 70/100)
divideParam.valRatio	розмір перевірочної вибірки відносно всіх початкових даних (приклад: 20/100)
divideParam.testRatio	розмір тестової вибірки відносно всіх початкових даних (приклад: 10/100)

Практична робота 6
Апроксимація функцій багатошаровим персептроном в середовищі MATLAB

Мета: Отримати практичні навички створення і навчання багатошарового персептрона в пакеті MatLab та створення файлів з даними, .

Завдання

1. Створити файл .csv в EXCEL з даними функції y=sin(x)/x.
2. Прочитати цей файл в MatLab за допомогою csvread.
3. Записати і прочитати файли з родільниками dlmread.
4. Створити й навчити нейронну мережу з трьох шарів: вхідний шар з 2-ма нейронами, прихований шар з двома нейронами і вихідний шар з одним нейроном (5-3) для апроксимації функції y=sin(x)/x
5. Проаналізувати отримані результати і зробити висновки.

Зміст звіту

1. Титульна сторінка.
2. Тема практичного заняття.
3. Мета роботи.
4. Завдання.
5. Опис виконання завдань по пунктам з наданням рисунків і скріншотів.
6. Текст програми MATLAB
7. Висновки.

Контрольні питання

1. Які знаки використовуються в якості роздільників?
2. Чим відрізняється функція csvwrite від dlmwrite?
3. Призначення валідаційної вибірки.

Теоретичні відомості

6.1. Робота з файлами CSV

MATLAB надає користувачам різні можливості; читання CSV є одним із варіантів у MATLAB. У форматі CSV дані розділені комами. Формат файлу CSV сумісний з різними типами програмних додатків і добре сприймається людиною. З цієї причини формат файлу CSV широко використовується, і MATLAB надає різні типи попередньо визначених функцій для читання записів із файлу CSV. У MATLAB є кілька варіантів (синтаксисів) читання файлів CSV.

Функція csvread – Зчитати файл із значеннями, розділеними комами.

Syntax:

M = csvread (FILENAME, M) зчитує матрицю M з файлу FILENAME як значення, розділені комами.

Приклад 1.

```
M=csvread('Cos_x.csv');
```

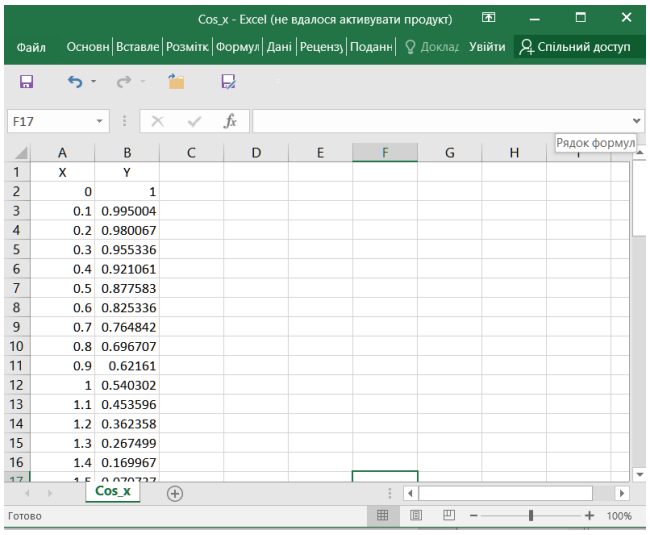
Розмір матриці визначається по кількості рядків і кількості колонок у файлі. Для даної команди файл Cos_x.csv повинен знаходитись в Current Folder, якщо файл знаходиться в іншій директорії, то треба вказати повний шлях до файлу.

M = csvread (FILENAME, M, R, C) зчитує матрицю M, починаючи зі зсуву рядок R і стовпець C у файлі до кінця файлу. R і C починаються з нуля, тому R=0 і C=0 визначає перше число у файлі.

Приклад 2.

```
M=csvread('Cos_x.csv', 1, 0);
```

У другому синтаксисі додатково надається значення зсуву для матриці. Це дає змогу зчитувати не всю матрицю до кінця, а тільки підматрицю від заданого номеру рядка і номера колонки до кінця файлу. Для даного прикладу це означає, що перший рядок пропускається. Це використовується, коли в першому рядку записані не дані (числа), а назви рядків. Файл Cos_x.csv містить 63 значення для функції y=Cos(x) (рис. 6.1).



	A	B
1	X	Y
2	0	1
3	0.1	0.995004
4	0.2	0.980067
5	0.3	0.955336
6	0.4	0.921061
7	0.5	0.877583
8	0.6	0.825336
9	0.7	0.764842
10	0.8	0.696707
11	0.9	0.62161
12	1	0.540302
13	1.1	0.453596
14	1.2	0.362358
15	1.3	0.267499
16	1.4	0.169967

Рисунок 6.1 – зміст файлу Cos_x.csv

M = csvread (FILENAME, M, R, C, [R1 C1 R2 C2])

зчитує підматрицю M, що задається верхнім правим кутом (**R1 C1**) і нижнім лівим кутом (**R2 C2**), початок даних задається **R, C**.

Приклад 3.

```
M=csvread('Cos_x.csv', 1, 0, [1 0 10 1]);
```

У третьому синтаксисі є можливість прочитати підматрицю не до кінця файлу, а в заданому діапазоні. У даному прикладі перший рядок пропускається і зчитується 2 колонки по 10 значень.

Функція csvwrite – записати файл із значеннями, розділеними комами.

csvwrite(FILENAME, M) записує матрицю M у FILENAME як значення, розділені комами.

csvwrite(FILENAME, M, R, C) записує матрицю M, починаючи зі зсуву рядок R і стовпець C у файлі. R і C починаються з нуля, тому R=0 і C=0 визначає перше число у файлі.

Примітки:

*csvwrite завершує кожен рядок символом переводу рядка без повернення каретки.

*csvwrite записує максимум п’ять значущих цифр. Для більшої точності, застосовується dlmwrite з аргументом точності.

Спеціалізовані файли

Наведені нижче функції належать до деяких спеціалізованих файлів.

• M = dlmread(filename, delimiter) зчитує дані з файлу filename з роздільником ASCII, використовуючи роздільник delimiter, в масив M. Використовуйте '\t', символ табуляції, як роздільник.

• M = dlmread(filename, delimiter, r, c) зчитує дані з файлу filename з роздільником ASCII, використовуючи роздільник delimiter, масив M, починаючи зі зміщенням r (по рядках) і c (по стовпцях). Параметри r та c відраховуються, починаючи з нуля, так що r=0, c=0 відповідає першому значення у файлі.

• M = dlmread(filename, delimiter, r, c, range) імпортує індексований або іменований діапазон даних з роздільниками у форматі ASCII. Для використання діапазону осередків необхідно визначити параметр

range у наступному виді:

range = [ВерхнійРядок, ЛівийСтовбець, НижнійРядок, ПравийСтовбець]

Аргументи функції dlmread такі: delimiter – символ, що відокремлює окремі матричні елементи в електронній таблиці формату ASCII, символ кома (,) – роздільник за замовчуванням, r, c – комірка електронної таблиці, з якої беруться матричні елементи, що відповідають елементам у верхньому лівому куті таблиці. range – вектор, що визначає діапазон комірок електронної таблиці.

Команда dlmwrite перетворює матрицю MATLAB на файл з ASCII роздільниками:

• dlmwrite(filename,A,delimiter) записує матрицю A у верхню ліву комірку електронної таблиці filename, використовуючи роздільник delimiter для відокремлення елементів матриці. Використовуйте '\t' для створення файлу із елементами, розділеними табуляцією. Всі елементи зі значенням 0 опускаються. Наприклад, масив [1 0 2] з'явиться у файлі у вигляді '1,2' (якщо роздільником є кома);

• dlmwrite(filename,A,delimiter,r,c) записує матрицю A у файл filename, починаючи з комірки, визначеної r і c, використовуючи роздільник delimiter.

Приклад. Створення і завантаження файлу даних з роздільником «табуляція»

```
clear all;
x=[0:.2:2*pi];
y=cos(x);
x1=x';
y1=y';
M=[x1,y1];
FName=('ccc.txt');
%Запис файлу;
dlmwrite(FName,M,'\t');
%Завантаження файлу в матрицю N з сувом на 1 рядок;
N = dlmread(FName,'\t',1,0);
```

6.2. Апроксимація функції нейронною мережею

Кількість нейронів вхідного шару відповідає кількості вхідних змінних мережі *X*.

Завданням нейронів цього шару є тільки розподіл вхідних сигналів по нейронах прихованого шару, підсумовування і обчислення функції активації в них не відбувається.

Кількість нейронів в прихованому шарі може бути різною і часто підбирається експериментально. Недостатня або надмірна кількість нейронів в прихованому шарі приводить до погіршення точності апроксимації. Крім того, надмірна кількість ускладнює мережу і зменшує швидкодію.

Нейрони вихідного шару формують вихідні сигнали, їх кількість відповідає кількості виходів *Y*.

Для навчання мережі складається навчальна вибірка вхідних сигналів і відповідних їм вихідних. Вибірка може бути розділена на три частини: робочу

вибірку (на основі якої проводиться навчання), валідаційну (для виявлення перенавчання мережі) і тестову вибірку (для перевірки якості навчання).

Приклад. Апроксимація функції Cos(x), поданої як .csv файл

```
clear all;
close all;
clc;
        %Читання файлу в матрицю M
M=csvread('Cos(x).csv');
  %Читання x1 з першого стовпця матриці M і перетворення у вектор
x1=M(:,1)';
    %Читання вектору y1 з другого стовпця матриці M
y1=M(:,2)';
%створення багатошарового персептрона
%схованих шарів – 2 у першому 5, у другому 3 – нейрона
% функція активації нейронів – logsig (сигмоїда)
%алгоритм навчання trainlm (Макварта-Левенберга)
net1=newff(x1,y1,[5,3],{'logsig','logsig'},'trainlm');
% Додаткові параметри навчання мережі
net1trainparam.lr= 0.1;     % коефіцієнт швидкості навчання
net1.trainparam.epochs=500;    % максимальна кількість епох навчання
net1.trainparam.show=25;    % кількість епох відображення на графіку
net1.trainparam. goal=0.0001;   % значення похибки функції втрат
         % параметри розподілу вхідної вибірки у відсотках
net1.divideParam.trainRatio = 70/100; % навчальна вибірка
net1.divideParam.valRatio = 20/100; % валідаційна вибірка
net1.divideParam.testRatio = 10/100;    % і тестова вибірка
       % навчання мережі
net1=train(net1, x1, y1);
% створення тестової вибірки з даними, які не увійшли до навчальної
% вибірки
b=[.05:.1:2*pi];
a1=sim(net1, b);  % перевірка навченої мережі на вибірці b
%Побудова графіка вхідної функції і апроксимованої
plot(x1,y1,'.',x1,a1,'r+');
```

Практична робота 7
Дослідження мережі Хопфілда

Мета роботи: отримати практичні навички рішення задачі розпізнавання образів з застосуванням нейромережі Хопфілда у пакеті MATLAB.

Завдання до роботи

1. Вивчити функцію **newhop()** на прикладі з 4 нейронами.
2. Згідно з варіантом (табл.7.1) створити мережу для розпізнавання трьох символів з файлу Alphabet.csv (аналог набору **prprob** MATLAB, символи A-Z).
3. Подати спотворені символи з завдання на мережу Хопфілда з різним ступенем спотворення, зробити висновки з точності розпізнавання.
4. Подати символ, що не увійшов в начальну вибірку, зробити висновки з результату розпізнавання.

Таблиця 7.1 – Варіанти завдань

№ з/п	Символи	№ з/п	Символи
1.	B, D, Y	11.	C, H,I
2.	F, U, M	12.	E, K, V
3.	N, D, X	13.	P, Q, Z
4.	L, O, S	14.	T, W, A
5.	G, J, Z	15.	S, T, O
6.	C, I, Z	16.	F, Y, S
7.	D, V, J	17.	Z, O, I
8.	N, P, A	18.	M, W, H
9.	K, L, B	19.	T, W, R
10.	R, C, Z	20.	Q, F, U

Зміст звіту

1. Мета роботи.
2. Скріпт-файл програми для пунктів 3, 4.
3. Прінт-скріни результатів дослідження мережі з різним ступенем спотворення символів.
4. Прінт-скріни розпізнавання символу, що не увійшов в навчальну вибірку.
5. Висновки.

Контрольні питання

1. Які типи задач розв’язуються мережею Хопфілда.
2. Умови стійкості мережі Хопфілда.
3. Недоліки мережі Хопфілда.
4. Призначення енергетичної функції.
5. Критерії завершення розпізнавання.

Теоретичні відомості

Структура мережі Хопфілда наведена на рис. 7.1.

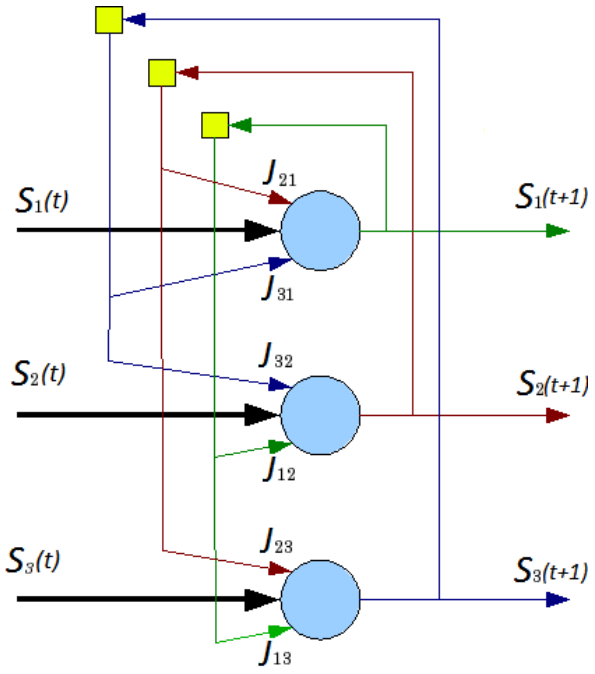


Рисунок 7.1 – Структурна схема мережі Хопфілда.

Слід зазначити, що в мережах зі зворотними зв'язками стан нейронів обчислюється доти, поки вони не виявляться сталими, не змінюваними згодом. Можна сказати, що мережа Хопфілда за певних умов збігається до сталого стану за скінченний час.

Енергетична функція

Використовується для оцінки стану мережі Хопфілда, що дозволяє оцінювати стани і обирати стани з найменшою енергією. Це може вказувати на те, що стан з найменшою енергією є найкращим (оптимальним) серед інших.

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} y_i y_j - \sum_{j=1}^{N} x_j y_j + \sum_{j=1}^{N} \theta_j y_j$$

Для енергетичної функції вхідні значення повинні бути 0, 1.

Мережа Хопфілда є автоасоціативною мережею. Дискретна мережа Хопфілда має наступні характеристики: вона містить один шар елементів, кожен елемент зв'язується з усіма іншими елементами, але не пов'язаний з самим собою; за один крок роботи оновлюється лише один елемент мережі; елементи оновлюються у випадковому порядку; вихід елемента обмежений значеннями 0 або 1.

У мережі Хопфілда вхідні сигнали нейронів є одночасно і вихідними сигналами мережі: $x_i(k) = y_i(k-1)$, при цьому збудливий вектор особливо не виділяється. У класичній системі Хопфілда відсутній зв'язок нейрона з власним виходом, що відповідає $w_{ij}=0$, а вся матриця ваг є симетричною: $w_{ij} = w_{ji}$

$$\hat{W} = \hat{W}^T$$

Симетричність матриці ваг гарантує збіжність процесу навчання. Дана мережа не використовує ані навчання з учителем, ані навчання без учителя. Вагові коефіцієнти в ній розраховуються тільки перед початком функціонування мережі на основі інформації про оброблювані дані, і все навчання мережі зводиться саме до цього розрахунку. Мережа фактично запам'ятовує образи до того, як на її вхід надходять реальні дані, і не може змінювати свою поведінку, тому говорити про зворотний зв'язок із учителем не доводиться.

Процес навчання мережі формує зони тяжіння деяких точок рівноваги, відповідних навчальним даним. При використанні мережі в якості асоціативної пам'яті маємо справу з навчальним вектором, або з множиною цих векторів, які в результаті проведеного навчання визначають розташування конкретних точок тяжіння (атракторів). Кожен нейрон має порогову (Signum) функцію активації зі значеннями ±1:

$$sign(a) = \begin{cases} 1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

Це означає, що вихідний сигнал *i*-го нейрона визначається функцією:

$$y_i = sign\left(\sum_{j=1}^{N} w_{ij} x_j + b_i \right)$$

де N позначає кількість нейронів. Часто постійна складова b_i, яка визначає поріг спрацьовування окремих нейронів, дорівнює 0. Тоді циклічне проходження сигналу в мережі Хопфілда можна надати співвідношенням:

$$y_i = sign\left(\sum_{j=1} w_{ij} y_j (k-1)) \right)$$

з початковою умовою $y_i\,(0)=x_i$.

У процесі функціонування мережі Хопфілда можна виділити два режими: навчання та класифікації. У режимі навчання на основі відомих навчальних вибірок обчислюються вагові коефіцієнти w_{ij}. У режимі класифікації при зафіксованих значеннях ваг і введенні конкретного початкового стану нейронів $y(0)=x$ виникає перехідний процес, що протікає відповідно з виразом (3) і закінчується в одному з локальних стійких станів, що задається біполярним вектором зі значеннями, $y=\pm 1$ для якого $y(k)=y(k-1)$.

Навчання не носить рекурентного характеру. Досить ввести значення (правило Хебба) ваг, визначивши їх через проекції вектору точки тяжіння еталонного образу:

$$w_{ij} = \frac{1}{N} x_i x_j$$

Згідно з цим правилом мережа дає правильний результат при вхідному векторі, що збігається з еталонним зразком, оскільки:

$$\sum_{j=1}^{N} w_{ij} x_j = \frac{1}{N} \sum_{j=1}^{N} x_i (x_j x_j) = x_i \sum_{j=1}^{N} 1 \frac{1}{N} = x_i$$

бо внаслідок біполярності значень елементів вектору x завжди $x_j^2 = 1$.

При введенні великої кількості навчальних вибірок $x^{(k)}$ для k=1, 2, ..., p ваги w_{ij} обчислюються згідно з узагальненим правилом Хебба відповідно до якого:

$$w_{ij} = \frac{1}{N}\sum_{k=1}^{l} x_i^{(k)} x_j^{(k)}$$

Завдяки такому режиму навчання ваги набирають значень, що є усередненням множини вибірок, пред'явлених до навчання. У разі декількох вибірок, актуальним стає питання про стабільність асоціативної пам'яті.

Після того як початкові умови задані у вигляді масиву T, що визначає ряд цільових вершин замкнутого гіперкуба, мережа для кожної вершини генерує вихід, який по зворотному зв'язку подається на вхід. Цей процес при створенні мережі повторюється багато разів, поки її вихід не перейде в стан рівноваги для кожної з цільових вершин. При подачі потім довільного вхідного вектора мережа Хопфілда переходить в результаті рекурсивного процесу до однієї з точок рівноваги, найбільш близької до вхідного сигналу.

Коли мережа Хопфілда спроектована, вона може бути перевірена з одним або більшим числом векторів входу. Досить імовірно, що вектори входу, близькі до цільових точок рівноваги, знайдуть свої цілі. Здатність мережі Хопфілда швидко обробляти набори векторів входу дозволяє перевірити мережу за відносно короткий час. Спочатку слід перевірити, що точки рівноваги цільових векторів дійсно належать вершинам гіперкуба, а потім можна визначити області тяжіння цих точок і виявити паразитні точки рівноваги (хибна пам'ять).

Створення нейронної мережі Хопфілда

Функция створення мережі Хопфілда має вигляд:

net=newhop(T),

де T – масив розміру R * Q, який об'єднує Q цільових векторів зі значеннями –1, +1 або 0, 1 для елементів; R – число елементів вектору входу.

Створимо нейронну мережу Хопфілда з чотирма нейронами і визначимо 4 точки рівноваги у вигляді наступного масиву цільових векторів:

```
T = [1 –1;  –1 1;  1 1;  –1 –1];
Покажемо точки рівноваги на графіку
T1 = T';
plot(T1(1,:), T1(2,:),'*r')
axis([-1.1 1.1 -1.1 1.1])
title('Точки рівноваги мережі Хопфілда')
xlabel('a(1)'), ylabel('a(2)')
```

На рис. 7.1 показані ці 4 точки рівноваги на площині станів мережі Хопфілда

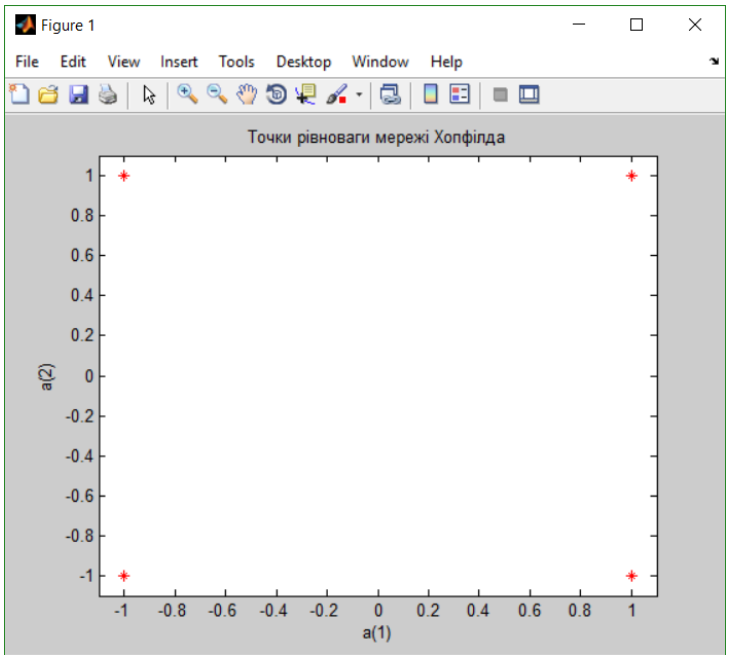


Рисунок 7.1 – Точки рівноваги на площині станів мережі Хопфілда

Розрахуємо ваги і зсуви мережі Хопфілда, скористаємось М-функцію newhop.

```
net = newhop(T1);
W = net.LW{1,1}
b = net.b{1,1}
W =
1.1618          0
     0     1.1618
b =
1.0e-016 *
          0
–0.1797
```

Перевіримо, чи належать вершини квадрата мережі Хопфілда:

```
Ai = T1;
[Y,Pf,Af] = sim(net, 4, [], Ai)
Y =
    1   -1    1   -1
   -1    1    1   -1
Pf =
    {}
Af =
   [2x4 double]
```

Як і слід було очікувати, виходи мережі дорівнюють цільовим векторам. Тепер перевіримо поведінку мережі при випадкових початкових умовах.

```
plot(T1(1,:), T1(2,:),'*r', 0,0,'rh'), hold on, axis([–1.1  1.1  –1.1  1.1])
xlabel('a(1)'), ylabel('a(2)')
new = newhop(T);
[Y,Pf,Af] = sim(net,4,[],T1);
```

```
for i = 1:20
    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,:))
end
```

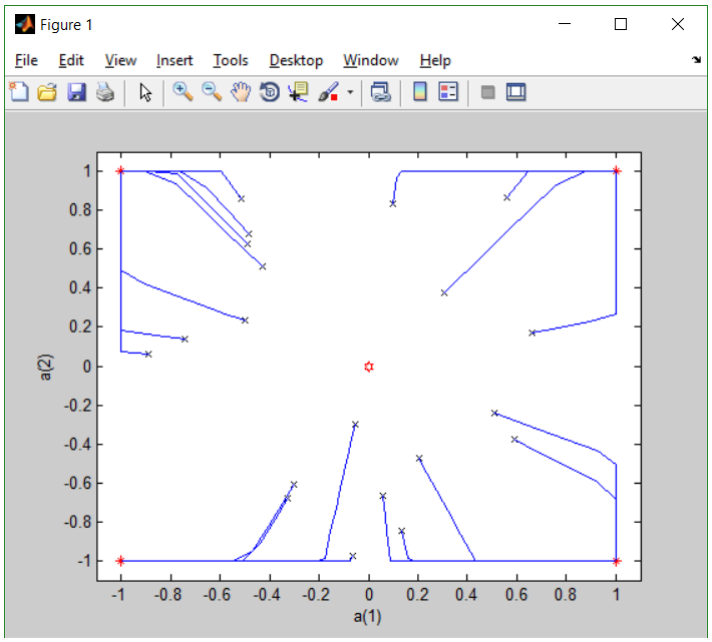


Рисунок 7.2 – Поведінка мережі при випадкових початкових умовах

Використання мережі Хопфіда в якості асоціативної пам’яті.

Завдання: створити і навчити мережу Хопфілда розпізнавати спотворені символи латинського алфавіту, дослідити вплив величини спотворень на точність розпізнавання. Скористаємось файлом Alphabet.csv, який містить матрицю 35x26 кодів великих літер латинського алфавіту. Кожна літера – це колонка, отримана з зображення літери розміром 5x7 елементів.

Приклад для символу A та J.

```
alphabet=csvread('Alphabet.csv');
L1 = alphabet(:, 1); %Завантаження літери A
L2 = alphabet(:, 10); %Завантаження літери J
%Виведення літери A J та у виді зображення 5x7 елементів
letterA = reshape(L1, 5, 7)';
letterA
subplot(1,4,1);  plotchar(L1);
letterA =
     0 0 1 0 0
     0 1 0 1 0
     0 1 0 1 0
     1 0 0 0 1
     1 1 1 1 1
     1 0 0 0 1
     1 0 0 0 1
```

```
%Створення мережі Хопфілда для розпізнавання літер A та J
T=[L1,L2];
net = newhop(T);
%Спотворення літер
noisyL1 = L1 + randn(35,1) *0.4;
noisyL2 = L2 + randn(35,1) *0.4;
%Виведення спотворених літер;
subplot(2,4,2); plotchar(noisyL1);
subplot(2,4,6); plotchar(noisyL2);
%Розпізнавання спотворених літер і виведення результату
[Y,Pf,Af] = sim(net, 1,{}, noisyL1);
subplot(2,4,3); plotchar(Y);
[Y1,Pf,Af] = net(1,{}, noisyL2);
subplot(2,4,7); plotchar(Y1);
%Друге розпізнавання спотворених літер і виведення результату
%Спотворюємо літеру A з коефіцієнтом 0.8
noisyL1 = L1 + randn(35,1) *0.8;
[Y2,Pf,Af] = net(1,{}, noisyL1);
subplot(2,4,4); plotchar(Y2);
%В якості спотвореної літери подаємо літеру V
noisyL3 = alphabet(:, 22);
[Y3,Pf,Af] = net(1,{}, noisyL3);
subplot(2,4,8); plotchar(Y3);
%Результати подано на рис. 7.3
```

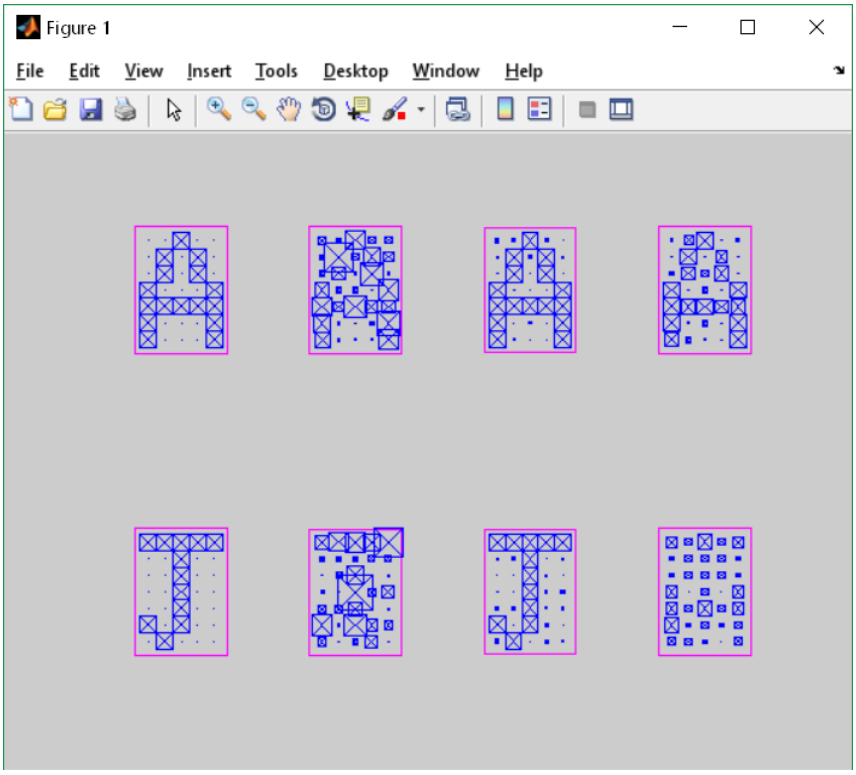


Рисунок 7.3 – Результат моделювання

Додаток А
Опис функцій MATLAB

Функція	Опис
B = reshape(A, m, n) або B = reshape(A,[m n])	Повертає матрицю B розміром m на n, елементи якої беруться по колонках з A. Якщо A не має m*, виникає помилка.
plotchar	Plot a 35 element vector as a 5x7 grid.
[Y,Pf,Af]=sim(net, {m n},{}, L) або [Y,Pf,Af]=net({m n}, {}, L)	Повертає матрицю Y розміром L x n з виходу мережі, n – кількість ітерацій (сходження к стійкому стану) мережі (подача виходу на вхід), L – вхідний образ (колонка). '. Вхідний образ L подається на вхід мережі, здійснюється 5 циклів ітерації, змінна B набуває значення другої ітерації. [Y, Pf, Af]=sim(net, {1 5},{}, L); B=Y{2};
T=csvread(FailMName)	Читання файлу даних формату .cvs (дані розділені комами) в змінну T. файл повинен бути в CurrentFolder MATLABб або вказується повний шлях до файлу. **Приклад.** alphabet=csvread('Alphabet.csv'); alphabet=csvread('C:\User\Documents\Alphabet.csv')

Практична робота 8
Дослідження генетичних алгоритмів на задачі пошуку екстремумів функції за допомогою засобів MATLAB

Мета. Вивчити основні принципи генетичного алгоритму та придбати навички оптимізації функцій за допомогою генетичних алгоритмів засобами MATLAB.

Завдання

1. Створити скріпт-файл оптимізації функції згідно з варіантом (табл. 8.1)
2. Побудувати графік функції.
3. Оптимізувати функцію з використанням функцій MATLAB
4. Оптимізувати функцію з використанням GUI інтерфейсу алгоритму.

Таблиця 8.1 – Функції для оптимізації.

№ з/п	Функція	Інтервал	Знайти
1.	$y(x) = 3\sqrt[3]{(x+4)^2} - 2x - 8$	[– 6; – 2]	максимум
2.	$y = \frac{\sqrt[3]{6(x-3)^2}}{(x-1)^2+8}$	[– 4; 8]	мінімум
3.	$z(x,y) = xe^{(-x^2-y^2)}$	[-2; 2]	мінімум
4.	$y = \frac{\sqrt[3]{6(x-3)^2}}{(x-1)^2+8}$	[– 4; 8]	максимум
5.	$z(x,y) = xe^{(-x^2-y^2)}$	[-2; 2]	максимум

Зміст звіту

1. Титульна сторінка.
2. Мета роботи.
3. Завдання.
4. Скріпт-файл оптимізації функцій.
5. Опис виконання по пунктам завдання (хід роботи) із скріншотами.
6. Висновки.

Контрольні питання

1. Перерахуйте основні особливості ГА.
2. Перелічіть генетичні оператори.
3. Які критерії зупинки використовуються для ГА?
4. Опишіть схему класичного ГА.
5. У чому полягають особливості спільного використання генетичних операторів?

Теоретичні відомості

8.1 Основи генетичних алгоритмів

Генетичні алгоритми (*Genetic Algorithms*) є складовою еволюційних методів, оскільки виникли в результаті спроб копіювання еволюції живих організмів. Генетичні алгоритми (ГА) – це процедури пошуку, засновані на механізмах природного відбору і спадкування; в ГА використовується принцип виживання найбільш пристосованих індивідів. ГА мають певні переваги в порівнянні з традиційними методами оптимізації, оскільки поєднують кращі властивості градієнтних методів та методів евристичного пошуку.

Уявімо штучний світ, населений кількістю N індивідів (особин), причому генетичний код кожного індивіду – це деякий розв'язок нашої задачі. Разом усі індивіди утворюють популяцію P_0. Для простоти вважається, що генетичний код кожного індивіда (генотип) записується у вигляді однієї хромосоми (*Chromosome*) X (для людини генотип містить 46 хромосом). У ГА хромосома X індивіда є впорядкованою послідовністю з M генів, тобто хромосома $X = \{G_1, G_2, ..., G_M\}$ – числовий вектор, який описує розв'язок задачі. Генотип усіх індивідів утворює популяцію хромосом $P = \{X_1, X_2, ... , X_N\}$. Для утворення нових хромосом і пошуку серед них найкращих використовуються оператори (методи): схрещування – об'єднання частин хромосом-батьків; мутації – випадкової зміни окремих генів; селекції – вибору хромосом для наступної ітерації алгоритму.

Основні відмінності ГА від традиційних задач оптимізації такі:

1) виконують пошук розв'язку виходячи не з однієї точки, а з деякої популяції;
2) використовують тільки цільову функцію, а не її похідні або іншу додаткову інформацію;
3) використовують ймовірнісні, а не детерміновані правила відбору.

Терміни ГА.

Популяція – кінцева множина особин.

Індивидіви, що входять в популяцію представляються хромосомами (бітовий рядок) з закодованими у них множинами параметрів задачі, точками в просторі пошуку (search points).

Хромосоми (ланцюжки, або кодові послідовності) – це впорядковані послідовності генів.

Ген (властивість, знак або детектор) – атомарний елемент генотипу, зокрема хромосоми.

Генотип – набір хромосом даного індивіду.

Фенотип – набір значень, що відповідають даному генотипу.

Алель – значення конкретного гена (властивості або варіанта властивості).

Локус – позиція гена в хромосомі (ланцюжку).

Покоління – чергова популяція в генетичному алгоритмі

Функція пристосованості (fitness function) – міра пристосованості даного індивіда в популяції. Задача оптимізації цільової функції зводиться до пошуку найбільш пристосованого індивіда.

Методи селекції хромосом

Метод рулетки.

У *методі рулетки* кожній хромосомі X_i ставиться у відповідність сектор колеса рулетки, величина якого пропорційна до функції пристосованості F даної хромосоми.

Ймовірність селекції хромосоми X_i, де $i = 1...N$, дорівнює:

$$v(x_i) = p_s(x_i)100\%$$

де:

$$p_s(x_i) = \frac{F(x_i)}{\sum_{j=1}^{N} F(x_j)}$$

У результаті селекції формується батьківська популяція з чисельністю N, в яку індивіди з великим значенням $p_s(X_i)$ можуть увійти кілька разів, а з малим значенням $p_s(X_i)$ – рідко.

Рангова селекція.

При *ранговій селекції* індивіди популяції впорядковуються за значенням їх функції пристосованості F (за спаданням), де кожній хромосомі X ставиться у відповідність її номер i у списку (ранг).

Ймовірність вибору хромосоми в такому випадку дорівнює:

$$p_s(x_i) = \begin{cases} 1/\mu, 1 \le i \le \mu \\ 0, \mu < i \le N \end{cases}$$

Турнірна селекція.

При *турнірному відборі* з популяції (N хромосом) випадковим способом вибирається t хромосом, краща з яких (переможець) записується у масив батьківських хромосом. Розмір туру $2 \le t < N$ за звичай дорівнює 2. Відбір повторюється, поки кількість батьківських хромосом не стане дорівнювати N.

Генетичні оператори

Оператор схрещування. На першому етапі схрещування вибираються пари хромосом із батьківської популяції. Операції схрещування полягають в обміні фрагментами ланцюжків між двома батьківськими хромосомами. Далі для кожної пари вибирається позиція гена (локус) у хромосомі, який визначає точку схрещування. Якщо хромосома містить L двійкових чисел, то точка схрещування L_K вибирається випадково з інтервалу [1, L]. У результаті схрещування утворюються два нащадки (рис. 8.1):

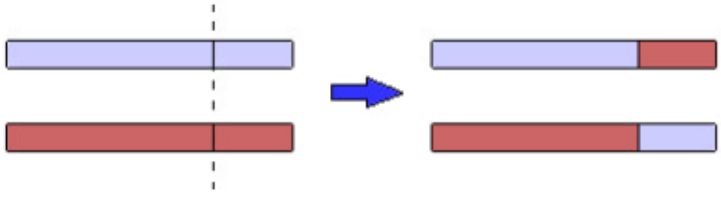


Рисунок 8.1 – Умовна схема схрещування

Оператор мутації з ймовірністю $p_м$ змінює значення гена на певну числову величину (амплітуду мутації A_M). В ГА мутація, імовірність якої звичайно дуже мала ($p_м < 0.1$), може виконуватися на популяції батьків перед схрещуванням або на популяції нащадків.

Класичний генетичний алгоритм

У виді блок-схеми алгоритм наведено на рис. 8.2.

У класичному ГА кожний ген записується у вигляді набору R бітів, тому вся хромосома записується набором двійкових чисел (алелів) довжиною $L = M$ (подібно до молекули ДНК). Кожен ген кодує певну властивість організму.

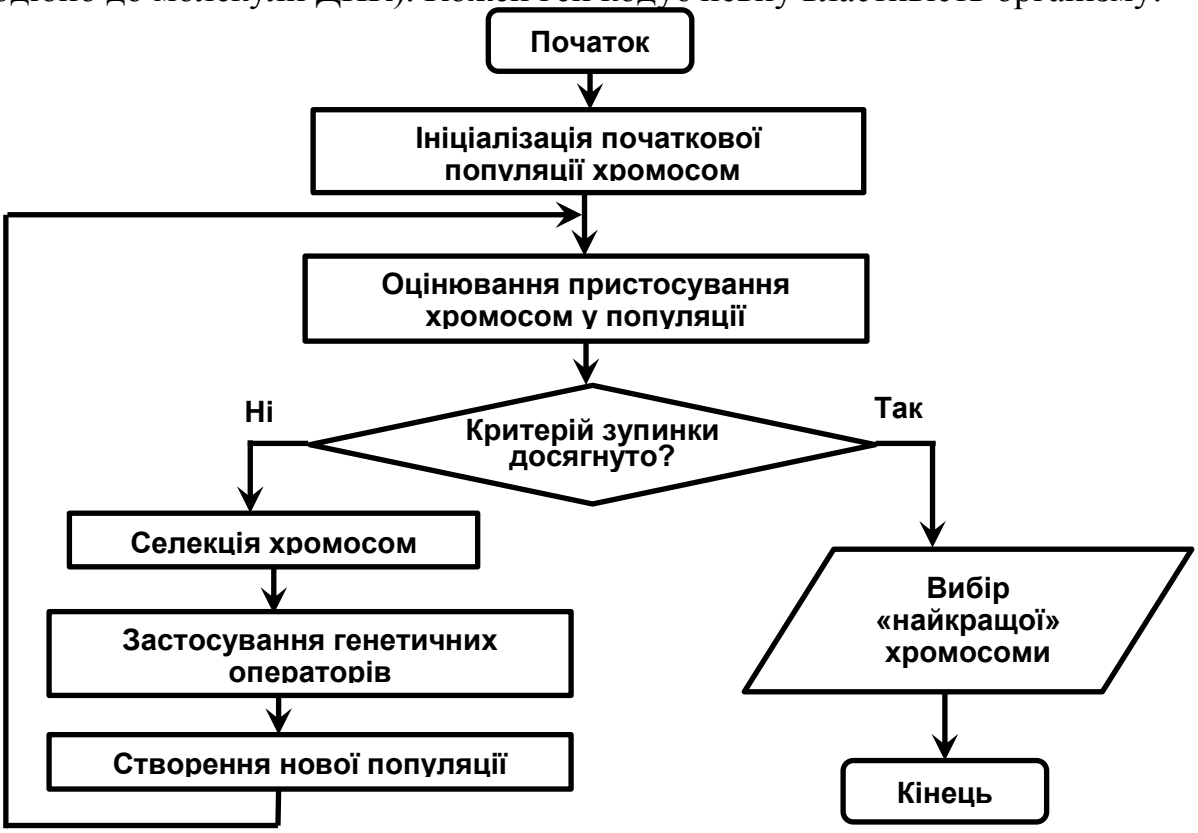


Рисунок 8.2 – Блок-схема класичного генетичного алгоритму

Також основний принцип роботи ГА можна описати за кроками.

1. Генеруємо початкову популяцію з n хромосом;
2. Обчислюємо кожної хромосоми її придатність;
3. Вибираємо пару хромосом-батьків за допомогою одного із способів відбору;
4. Проводимо схрещування (кросинговер) двох батьків із ймовірністю p_c, створюючи двох нащадків;
5. Проводимо мутацію нащадків із ймовірністю p_m;
6. Повторюємо кроки 3-5, доки не буде згенеровано нове покоління популяції, що містить n хромосом;
7. Повторюємо кроки 2-6, доки не буде досягнутий критерій закінчення процесу.

Основними параметрами ГА є:

- ймовірність мутації;
- точність отримання результату;

- кількість ітерацій алгоритму або кількість поколінь;
- обсяг популяції.

8.2 Генетичний алгоритм в MATLAB

Параметри ГА можна задавати командами або через графічний інтерфейс (GUI) генетичного алгоритму.

Приклад мінімізації з використанням команд

Розглянемо основні параметри ГА, які можна встановлювати в командному режимі до запуску алгоритму. При командному способі виклик ГА має такий вид:

[x, fval, exitflag, output, population, score] = ga(fitnessFunction, nvars, options);

де: x – знайдене значення аргументів функції, якому відповідає мінімум функції;

fval – значення знайденого мінімуму функції;

exitflag – ознака причини закінчення алгоритму;

output – короткий опис результатів алгоритму;

population – остання популяція хромосом (значення аргументів функції);

score – значення функції пристосованості для хромосом останньої популяції.

ga – ідентифікатор генетичного алгоритму (m-файл в MATLAB);

fitnessFunction – ідентифікатор m-файлу з функцією, що мінімізується;

nvars – кількість аргументів функції;

options – параметри алгоритму, що не задані за замовчуванням.

Основні параметри алгоритму.

Для задання функції, що мінімізується використовується команда:

fitnessFunction = @<FaleName>;

Приклад: fitnessFunction = @fit_fun,

де fit_fun – ім’я m-файлу з функцією, що мінімізується.

Параметри популяції

'PopulationSize', <кількість індивідів в популяції>

Параметри оператора селекції (відбору)

'SelectionFcn', <ідентифікатор методу селекції>

Ідентифікатори основних методів селекції:

@selectionroulette – рулетка;

@selectiontournament – турнірна селекція

@selectionuniform – батьки вибираються випадковим чином згідно з заданим розподілом та з урахуванням кількості батьківських особин та їх ймовірностей;

Параметри оператора мутації

Цей оператор необхідний для «вибивання» популяції з локального екстремуму та перешкоджає передчасній збіжності. Це досягається за рахунок того, що змінюється випадково вибраний ген у хромосомі.

'MutationFcn', {<ідентифікатор типу мутації>}

Ідентифікатори основних типів мутації:

@mutationgaussian – додає невелике випадкове число (відповідно до розподілу Гауса) до всіх компонентів кожного вектора-індивідіви

@mutationuniform – вибираються випадковим чином компоненти векторів і замість них записуються випадкові числа допустимого діапазону;

@mutationadaptivefeasible генерує набір напрямків залежно від останніх найбільш вдалих і невдалих поколінь і з урахуванням обмежень, що накладаються, просувається вздовж усіх напрямків на різну довжину;

Параметри оператора схрещування (кроссовер)

Кросовер – це операція, коли з двох хромосом породжується одна чи кілька нових хромосом. При цьому хромосоми розрізаються у випадковій точці та обмінюються частинами між собою.

'CrossoverFcn'{<ідентифікатор типу кроссовера>}

Ідентифікатори основних типів кроссовера:

@crossoversinglepoint – одноточковий

@crossovertwopoint – двохточковий

@crossoverarithmetic – арифметичний

@crossoverscattered – генерується випадковий двійковий вектор відповідності батьків

Умови зупинки алгоритму:

Generations – алгоритм зупиняється, коли кількість поколінь (ітерацій) досягає значення Generations.

TimeLimit – алгоритм зупиняється після закінчення певного заданого часу в секундах Time limit.

FitnessLimit – алгоритм зупиняється тоді, коли значення функції пристосованості для найкращої точки поточного популяції буде менше або дорівнювати Fitness limit.

StallGenLimit – алгоритм зупиняється у разі, якщо немає поліпшення для цільової функції у послідовності наступних популяцій один за одним довжиною StallGenLimit.

TolFun – алгоритм зупиняється, якщо середня відносна зміна найкращого значення функції придатності протягом поколінь StallGenLimit менше або дорівнює TolFun. Якщо StallTest є «geometricWeighted», тоді алгоритм зупиняється, якщо середньозважена відносна зміна менше або дорівнює TolFun.

Приклад задання параметрів ГА в скріпт-файлі.

```
        % GA options structure.
fitnessFunction = @ex13;       % Fitness function
nvars = 2;                     % Number of Variables
     % Start with default options
options = gaoptimset;
       % Modify some parameters
options = gaoptimset(options,'PopInitRange',[-4 ; -1 ]);
```

```
options = gaoptimset(options,'PopulationSize',10);
options = gaoptimset(options,'Timelimit', 100);
options = gaoptimset(options,'Fitnesslimit', -10000000);
options = gaoptimset(options,'Generations', 100);
options = gaoptimset(options,'CrossoverFcn',{@crossoverarithmetic});
options = gaoptimset(options,'MutationFcn',{@mutationgaussian 1 1});
options = gaoptimset(options,'Display','off');
options = gaoptimset(options,'SelectionFcn',@selectionroulette);
options = gaoptimset(options,'PlotFcns',{@gaplotbestf @gaplotbestindiv .... 
                    @gaplotdistance });
% Run GA
[X,FVAL,REASON,OUTPUT,POPULATION,SCORES]=ga(fitnessFunction,nvars,opti
ons);
```

Значення змінних X, FVAL, REASON, OUTPUT, POPULATION, SCORES можна подивитись в робочому полі MATLAB або вивести на екран командою disp().

Приклад мінімізації з використанням GUI-інтерфейсу

Приклад 1. Мінімізувати функцію одного аргументу:

$$f(x) = \sqrt{3 + x^2} + 3 * \cos(x)$$

Створимо m-file для даної функції і збережемо його в робочій директорії MATLAB під ім'ям **fit_fun.m**.

```
function y=fit_fun(x)
y=(3+x.^2.).^(1/2)+3.*cos(x);
end
```

Для початку роботи з GUI інтерфейсом необхідно набрати в командному вікні MATLAB: *optimtool* і натиснути Enter або на головному вікні MATLAB натиснути вкладку APPS і вибрати застосунок Optimization. З'явиться панель роботи оптимізаційного пакету Optimization Tools (рис. 8.3.) до якого входить генетичний алгоритм (ГА).

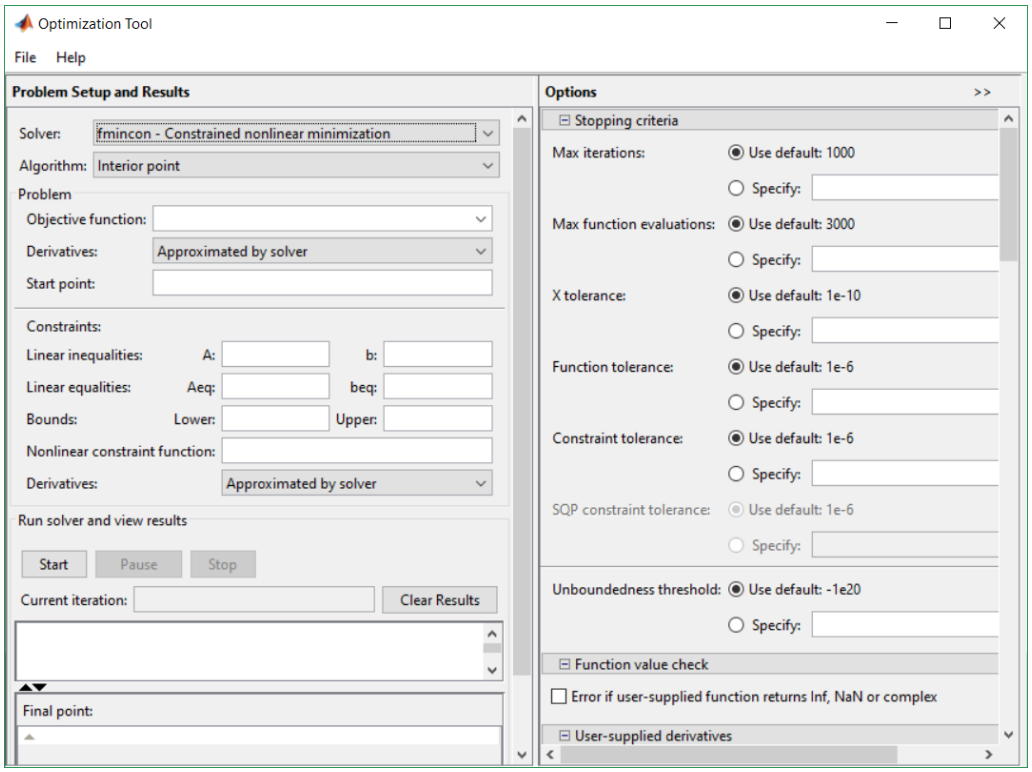


Рис 8.3 – Панель Optimization Tools

В полі Solver обрати ga – Genetic Algorithm (рис. 8.4)

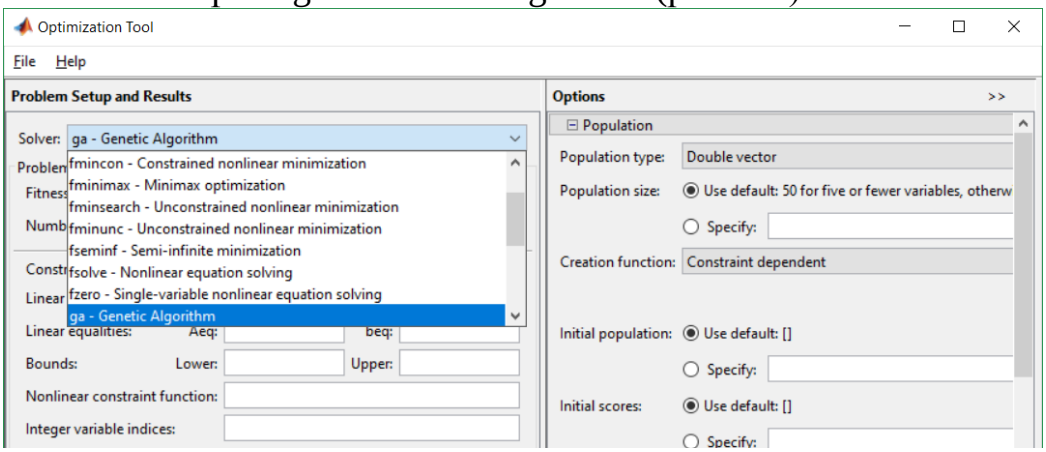


Рис 8.4. – Вибір серед методів оптимізації функції

З’явиться панель генетичного алгоритму (рис. 8.5.)

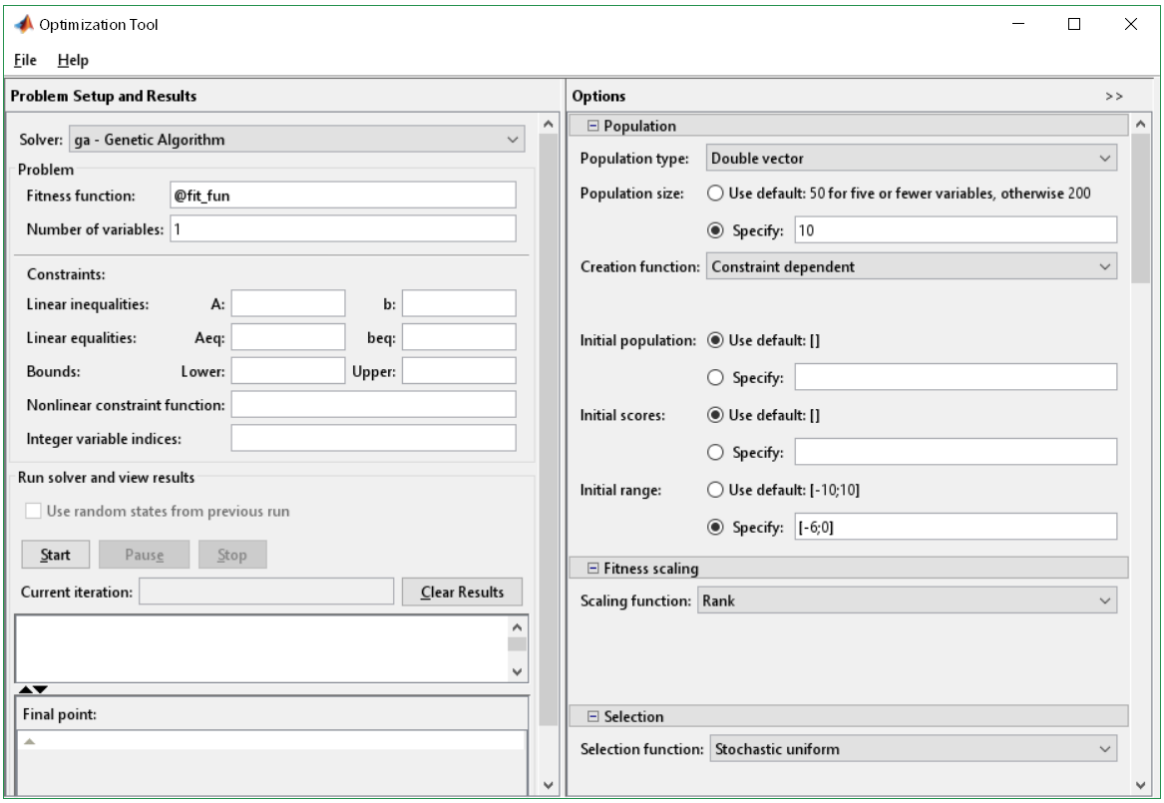


Рисунок 8.5 – Панель генетичного алгоритму

У вікні **Problem Setup and Results** введемо інформацію про функцію для оптимізації

В полі **Fitness function:** введемо ім’я цільової функції @fit_fun

В полі **Number of variables:** визначимо кількість змінних даної функції – 1.

Визначимо параметри ГА (вікно **Options**).

У підвікні **Populations** встановимо значення параметрів ГА:

В полі **Populations size (Specify):** кількість особин в популяції 10,

В полі **Initial range(Specify):** початковий інтервал, на якому буде здійснюватися пошук мінімуму функції = [-6; 0].

В пілвікні **Stopping criteria** (в нижній частині вікна **Options**)

В полі **Generations (Specify):** кількість поколінь = 100.

У підвікні **Plot functions** встановимо прапорці для **best fitness, best individual, distance.**

Клацнемо по кнопці **Start** у вікні **Problem Setup and Results.**

В результаті завершення процесу (рис. 8.6.) 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.

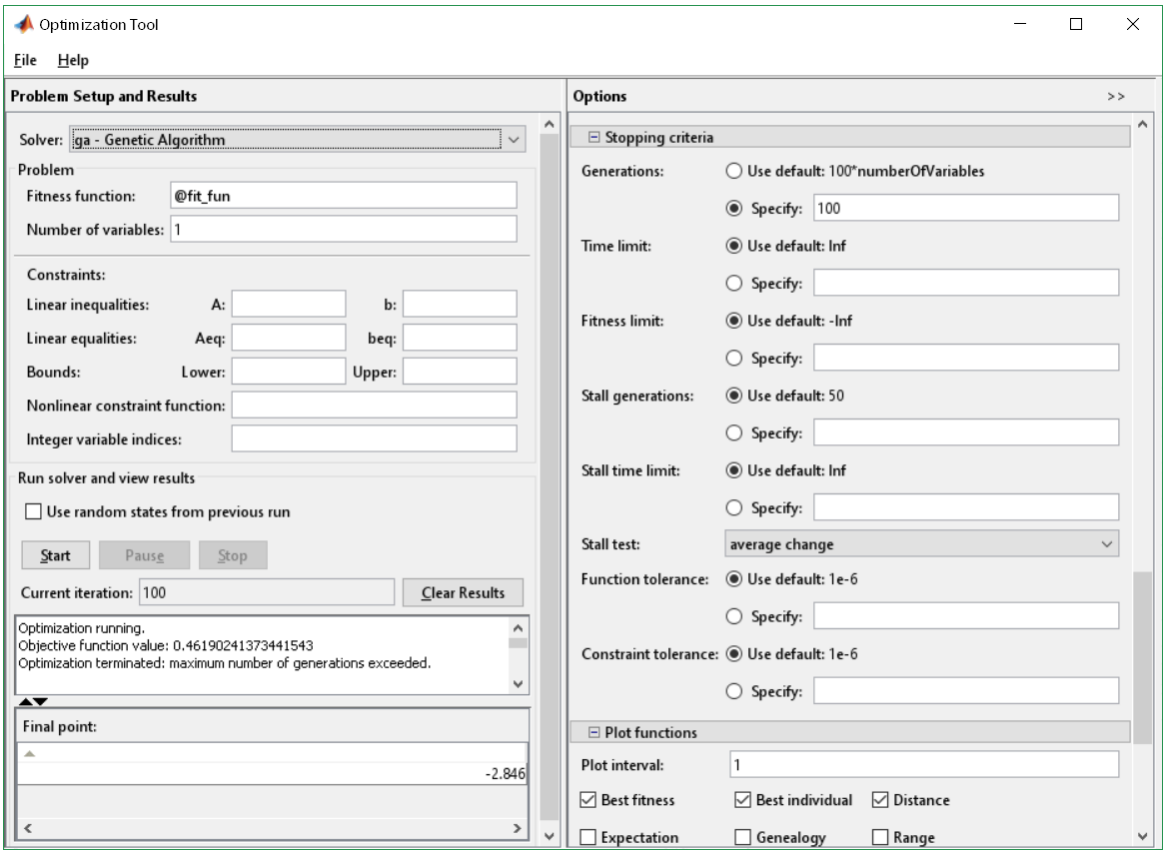


Рисунок 8.6 – Результат ГА для функції $f(x) = \sqrt{3 + x^2} + 3 * \cos(x)$ (fit_fun).

Для даної задачі результати вийшли наступні мінімум функції досягається в точці x = –2.846 і $f(-2.846) = 0.4619024137344$.

Процес пошуку можна спостерігати на графіках best fitness, best individual, distance (рис.8.7).

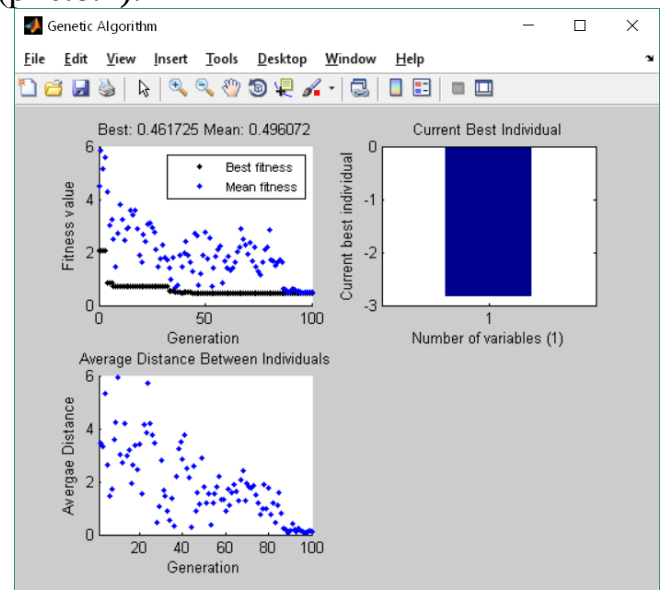


Рисунок 8.7 – Графічний аналіз рішення

Перший малюнок відображає зміну значення цільової функції. Видно, що, починаючи з 90 популяції, алгоритм зійшовся до вирішення. На другому малюнку зображена найкраща особина. Третій малюнок відповідає зміні відстані між особинами в поколіннях. Особи стають однаковими (хеммінгова відстань = 0) в останніх 10 поколіннях. ГА потрібно запустити кілька разів, а

потім вибрати оптимальне рішення. Це пов'язано з тим, що початкова популяція формується з використанням генератора випадкових чисел.

Переконатися в правильності рішення можна, побудувавши графік функції (рис. 8.8).

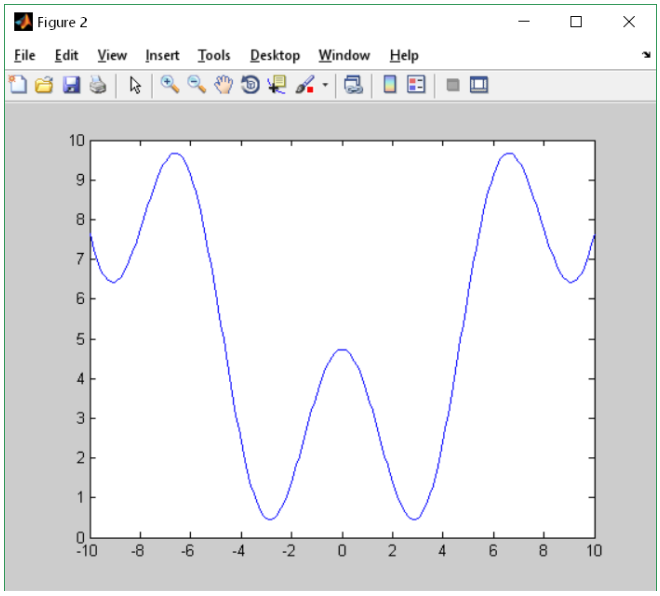


Рисунок 8.8 – Графік функції $f(x) = \sqrt{3+x^2} + 3*\cos(x)$

З графіка видно, що функція має 2 локальних мінімуми, тому при кількох запусках алгоритму можуть бути різні результати. Враховувати це можна задаючи алгоритму різні межі зміни аргументу функції. Для MATLAB версії 2022b при використанні вкладки `APPS` → `Optimization` з'явиться панель (рис. 8.9):

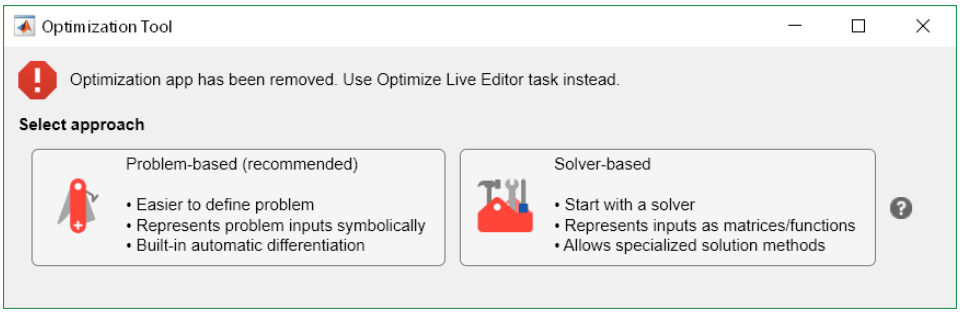


Рисунок 8.9 – Головна панель застосунку `Optimization`

Для роботи з конкретним методом оптимізації треба натиснути кнопку Solver-based (праворуч) для виклику редактора Live Editor (рис.8.10). Редактор пропонує, як приклад, оптимізацію функції з двома аргументами.

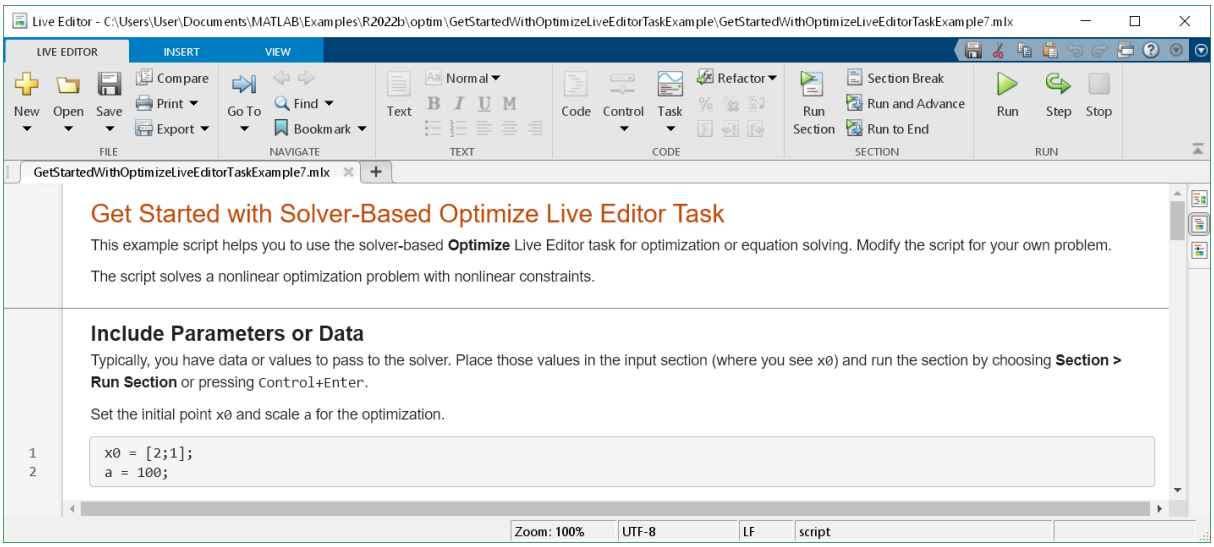


Рисунок 8.10 – Панель редактора Live Editor

Для подальшої роботи в командному вікні якій-небудь змінній присвойте значення 1 для оптимізації функції з одним аргументом або 2 для функції з двома аргументами, наприклад:

>> n=1;

Замініть у прикладі функцію `function f=objectiveeFcn(x,a)` на `function y=fit_fun(x)` або добавте її (рис.8.11)

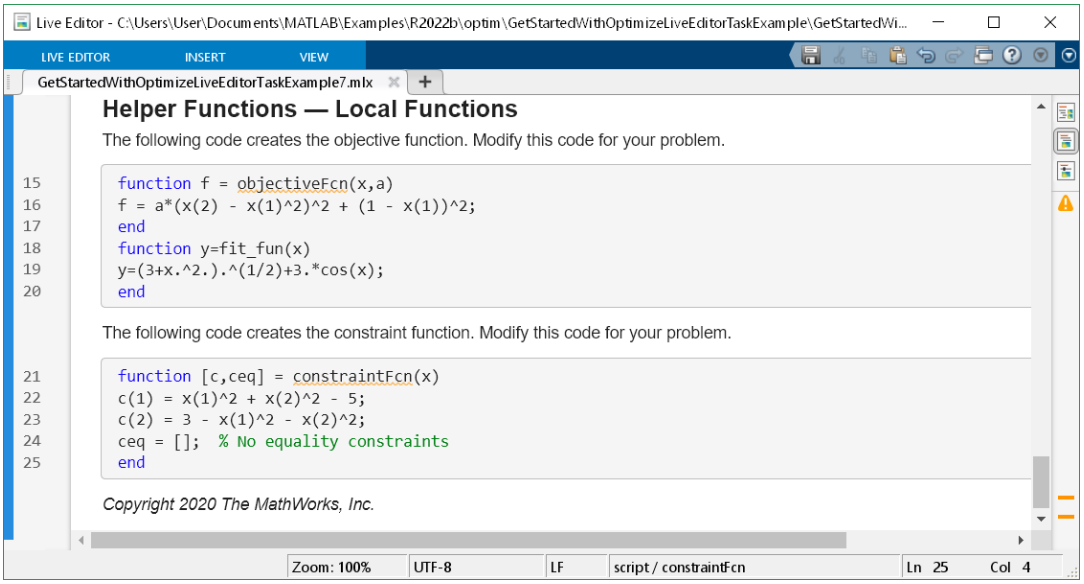


Рисунок 8.11 – Панель редактора Live Editor

Перейдіть до розділу Optimize (рис.8.12). У полі Specify problem type активуйте, якщо не активовані вікна Nonlinear і Unconstrained, у полі Solver виберіть ga – Genetic algorithm. У полі Objective function активуйте Local function і виберіть функцію fit_fun. У полі Number of variables виберіть n (кількість аргументів функції). У полі Plot увімкніть прапорці Distance, Best individual, Best Function. В меню редактора натисніть Run.

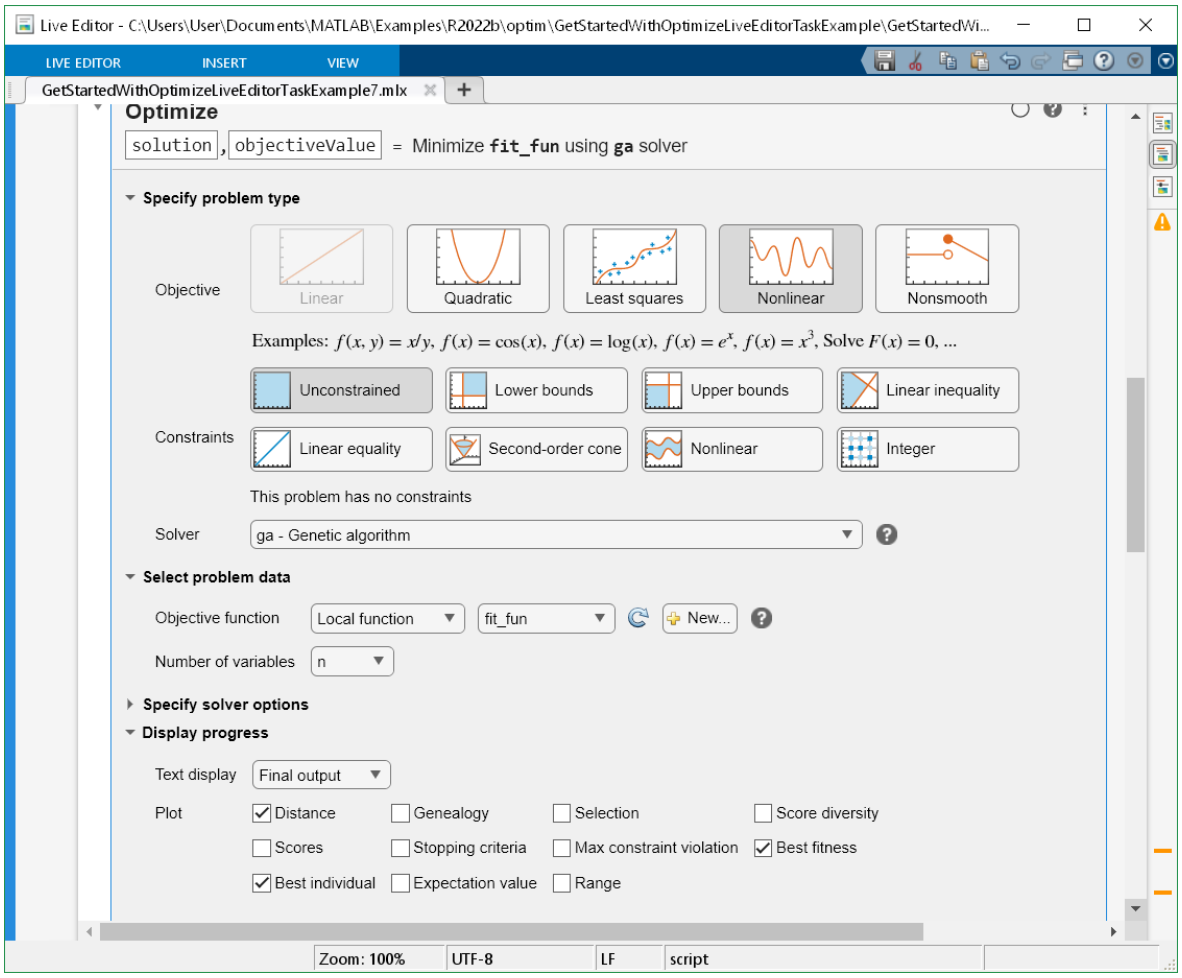


Рисунок 8.12 – Панель редактора Live Editor для оптимізації.

Результат оптимізації можна подивитись в розділі Results.

Приклад 2. Максимізувати функцію двох змінних**:**

$$z(x, y) = \exp(-x^2 - y^2) + \sin(x + y).$$

ГА вирішує тільки завдання мінімізації. Для знаходження максимуму функції *f(x)* слід мінімізувати функцію *–f(x)*.

Створимо M-file для функції *z(x) = –f(x)* і збережемо його в робочій директорії MATLAB під ім'ям ex13.m:

```
function z = ex13(x)
z = – (exp (–x (1).^2–x (2).^2) + sin (x (1) + x (2)));
end;
```

Перейдемо у робоче вікно ГА.

В полі **Fitness function:** введемо ім'я цільової функції @ex13

В полі **Number of variables:** визначимо кількість змінних даної функції – 2.

Визначимо параметри ГА (вікно **Options**).

У підвікні **Populations** встановимо значення параметрів ГА:

В полі **Populations size (Specify):** кількість особин в популяції 10,

В полі **Initial range(Specify):** початковий інтервал, на якому буде здійснюватися пошук мінімуму функції = [-1; 3].

В пілвікні **Stopping criteria** (в нижній частині вікна **Options**)

В полі **Generations (Specify):** кількість поколінь = 100.

У підвікні **Plot functions** встановимо прапорці для **best fitness, best individual, distance.**

Клацнемо по кнопці **Start** у вікні **Problem Setup and Results.**

В результаті завершення процесу (рис. 8.13.)

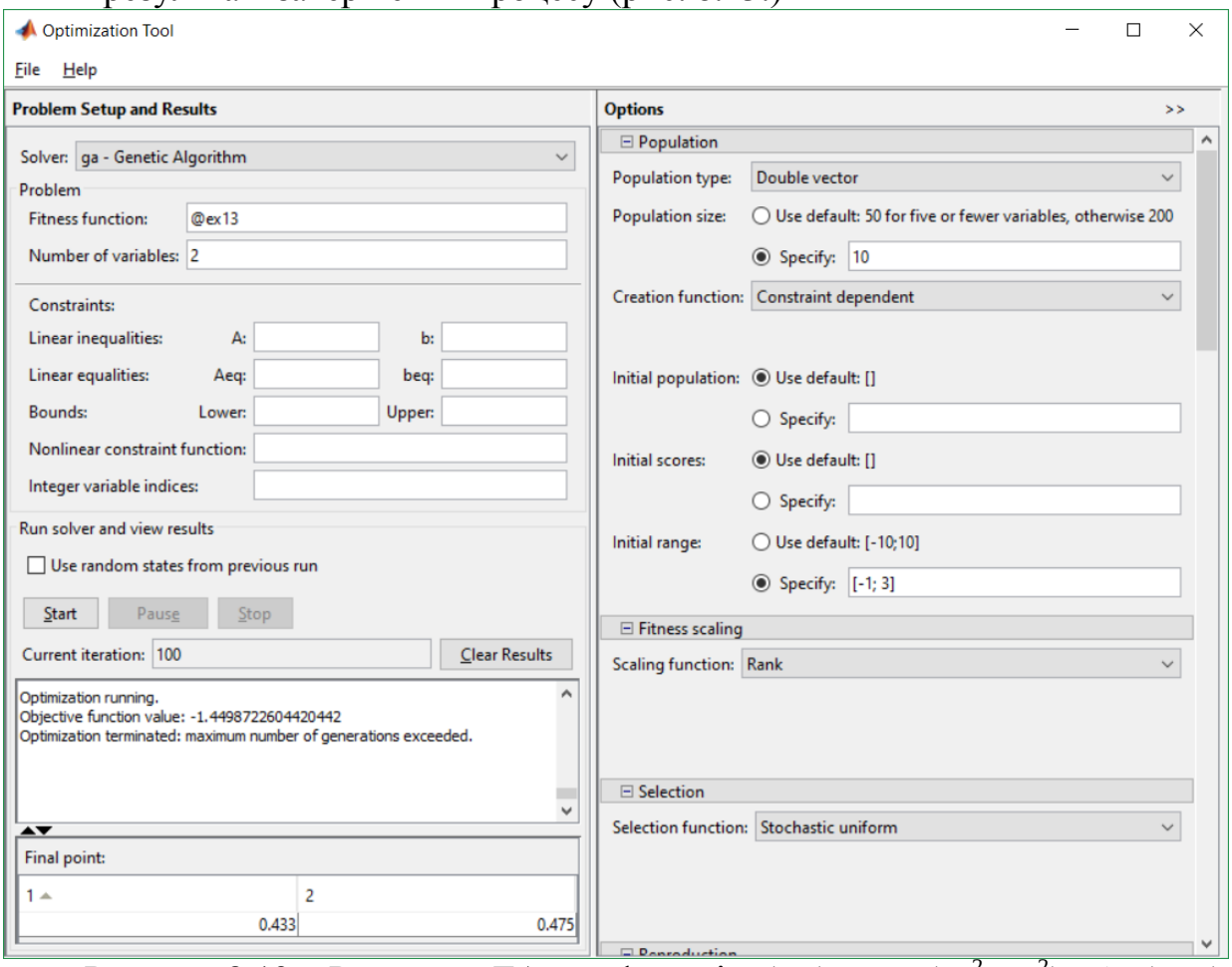


Рисунок 8.13 – Результат ГА для функції $z(x,y) = exp(–x^2 – y^2)+sin(x+y)$

У вікні **Final point** з'явиться значення змінних x, y відповідне мінімуму функції z, а у вікні **Status and result** можна побачити знайдене мінімальне значення цільової функції.

Для даної задачі результати вийшли наступні мінімум функції досягається в точці $x = 0.433$, $y=0.475$ і $z(0.433, 0.475) = –1.4498722604420442$.

Для контролю отриманих результатів побудуємо графік функції (рис. 8.10). з графіка можна зробити висновок, що максимум функції приблизно дорівнює 1.1 при x1 = 0.4, x2=0.4. Для побудови графіка скористуємось кодом:

```
[X1,X2]=meshgrid(-1:.1:3,-1:.1:3);
>> Z = (exp(-((X1).*(X1))-((X2).*(X2))+sin(X1+X2)));
%Z=ex13(x);
surf(X1,X2,Z);
```

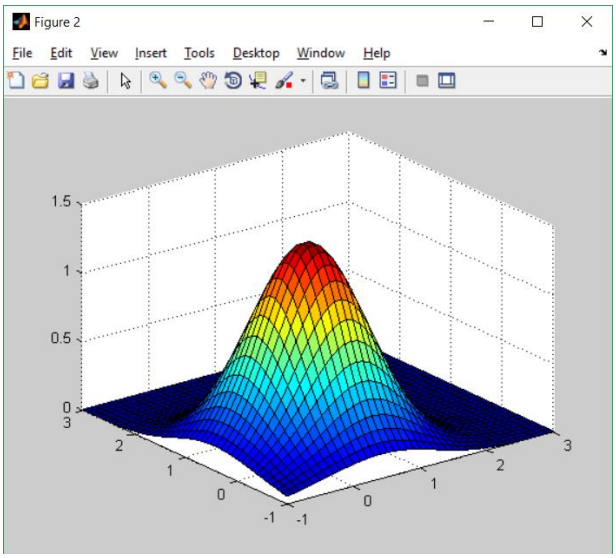


Рисунок 8.10 – Поверхня функції $z(x,y) = exp(–x^2 – y^2)+sin\ (x+y)$

Додаток А

Основні параметри ГА

Формат параметру

options = gaoptimset('param1', value1, 'param2', value2,...)

де: OptionName – ідентифікатор (рядок), значення параметру

Param	опис	приклад
PopInitRange	Інтервал зміни індивідів	options= gaoptimset('PopInitRange',[-4 ; -1]);
PopulationSize	Розмір популяції	options = gaoptimset('PopulationSize',10);
Timelimit'	Завершення алгоритму через заданий час (сек)	options = gaoptimset(options,'Timelimit', 100);
		options = gaoptimset('Fitnesslimit', -10000000);
Generations	Кількість ітерацій алгоритму	options = gaoptimset('Generations', 100)
SelectionFcn	Метод селекції	options = gaoptimset('SelectionFcn',@selectionroulette);

Література

1. Bratko Ivan Prolog programming for artificial intelligence. 3-d edition, Addison Wesley, Harlow [u.a.], 2010

2. SWI-Prolog Wiki facilities [Електронний ресурс]: [Веб-сайт] – Електронні дані. Режим доступу: https://www.swi-prolog.org/wiki/ – Назва з екрана.

3. Mark J. Wierman An Introduction to the Mathematics of Uncertainty, Creighton University, 2010, 367p.

4. J.-S. Roger Jang Ned Gulley MATLAB Fuzzy Logic Toolbox, User's Guide Fuzzy Logic Version 1, The MathWorks, Inc., 1997 208p.

5. Хоцкіна В.Б., Вдовиченко І.Н. Робота в пакеті MATLAB: Навчальний посібник. – Кривий Ріг: Державний університет економіки і технологій, 2023. – 130 с.

6. MATLAB в інженерних розрахунках. Комп'ютерний практикум : навч. посіб. / Н. М. Гоблик, В. В. Гоблик ; Нац. ун-т "Львів. політехніка". – 3-тє вид., допов. – Львів : Вид-во Львів. політехніки, 2020. – 191 с. : рис., табл. – Бібліогр.: с. 188.

7. Howard Demuth Mark Beale Neural Network Toolbox For Use with MATLAB User's Guide Version 4, The MathWorks, Inc., 2004, 846p.

8. Ямпольський Л.С. Нейротехнології та нейрокомп'ютерні системи: підручник / Л.С. Ямпольський, О.І. Лісовиченко, В.В. Олійник. – К.: «Дорадо-Друк», 2016. – 576 с.: іл.

9. Optimization Toolbox: Solve linear, quadratic, conic, integer, and nonlinear optimization problems [Електронний ресурс] https://www.mathworks.com/products/optimization.htmlp, назва з екрана.

10. Кононюк А.Ю. Нейронні мережі і генетичні алгоритми – К.: «Корнійчук», 2008. – 446 с.

Надруковано в РВЦ
Державного університету інформаційно-комунікаційних технологій
Формат 60х90/16. Папір друкарський.
Наклад 100 прим. Зам. 534.

Свідоцтво суб’єкта видавничої справи ДК №7917 від 16.08.2023 р.

03110, м. Київ, вул. Солом’янська, 7.
Тел. (044) 249-25-76