МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ
УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

Кафедра комп’ютерних систем та мереж

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт
з дисципліни

«Математичне забезпечення комп’ютерних систем та мереж»

для студентів денної та заочної форми навчання
за спеціальніст
123 «Комп’ютерна інженерія»

Тернопіль - 2024

Методичні вказівки розроблені у відповідності з навальним планом спеціальності 123 «Комп’ютерна інженерія».

Укладачі: Стадник Н.Б., к.т.н, старший викладач кафедри комп’ютерних систем та мереж

Рецензент: Готович В.А, к.т.н, доцент, доцент кафедри комп’ютерних наук

Затверджено на засіданні кафедри комп’ютерних систем та мереж, протокол №2 від 27 серпня 2024р.

Схвалено та рекомендовано до друку методичною комісією факультету комп’ютерно-інформаційних систем і програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя, протокол №1 від 2 вересня 2024р.

Методичні вказівки складені з урахуванням методичних розробок інших закладів вищої освіти, а також матеріалів літературних джерел, наведених у переліку.

ЗМІСТ

ВСТУП

Мета дисципліни «Математичне забезпечення комп'ютерних систем та мереж» полягає у отриманні необхідних теоретичних знань та практичних навичок з математичних основ комп'ютерних систем та мереж. Вивчення дисципліни дозволяє розвинути творчий потенціал, необхідний для самостійної постановки нових технічних завдань та пошуку їх рішень.

Предметом вивчення навчальної дисципліни є основні принципи машинного навчання та їхнє використання в системах штучного інтелекту, нейронні мережі, введення до експертних систем.

Завданням вивчення навчальної дисципліни є формування у студентів системного мислення, навичок використання математичних та програмних засобів для розробки систем штучного інтелекту різного типу та призначення.

Вивчення навчальної дисципліни передбачає формування та розвиток у студентів компетентностей:

Загальні:

ЗК1. Здатність до адаптації та дій в новій ситуації.

ЗК2. Здатність до абстрактного мислення, аналізу і синтезу.

ЗК3. Здатність проводити дослідження на відповідному рівні.

ЗК4. Здатність до пошуку, оброблення та аналізу інформації з різних джерел.

ЗК5. Здатність генерувати нові ідеї (креативність).

ЗК6. Здатність виявляти, ставити та вирішувати проблеми.

ЗК7. Здатність приймати обґрунтовані рішення.

Спеціальні (фахові):

СК2. Здатність розробляти алгоритмічне та програмне забезпечення, компоненти комп'ютерних систем та мереж, Інтернет додатків, кіберфізичних систем з використанням сучасних методів і мов програмування, а також засобів і систем автоматизації проектування.

СК4. Здатність будувати та досліджувати моделі комп'ютерних систем та мереж.

СК6. Здатність використовувати та впроваджувати нові технології, включаючи технології розумних, мобільних, зелених і безпечних обчислень, брати участь в модернізації та реконструкції комп’ютерних систем та мереж, різноманітних вбудованих і розподілених додатків, зокрема з метою підвищення їх ефективності.

СК9. Здатність представляти результати власних досліджень та/або розробок у вигляді презентацій, науково-технічних звітів, статей і доповідей на науково-технічних конференціях.

СК10. Здатність ідентифікувати, класифікувати та описувати роботу програмно-технічних засобів, комп’ютерних систем, мереж та їхніх компонентів.

СК11. Здатність обирати ефективні методи розв’язування складних задач комп’ютерної інженерії, критично оцінювати отримані результати та аргументувати прийняті рішення.

СК13. Здатність розробляти інтелектуалізовані системи опрацювання даних з використанням технологій інженерії знань та машинного навчання.

Програмні результати навчання:

РН1. Застосовувати загальні підходи пізнання, методи математики, природничих та інженерних наук до розв’язання складних задач комп’ютерної інженерії.

РН2. Знаходити необхідні дані, аналізувати та оцінювати їх.

РН3. Будувати та досліджувати моделі комп’ютерних систем і мереж, оцінювати їх адекватність, визначати межі застосовності.

РН4. Застосовувати спеціалізовані концептуальні знання, що включають сучасні наукові здобутки у сфері комп’ютерної інженерії, необхідні для професійної діяльності, оригінального мислення та проведення досліджень, критичного осмислення проблем інформаційних технологій та на межі галузей знань.

РН5. Розробляти і реалізовувати проекти у сфері комп’ютерної інженерії та дотичні до неї міждисциплінарні проєкти з урахуванням інженерних, соціальних, економічних, правових та інших аспектів.

РН6. Аналізувати проблематику, ідентифікувати та формулювати конкретні проблеми, що потребують вирішення, обирати ефективні методи їх вирішення.

РН7. Вирішувати задачі аналізу та синтезу комп’ютерних систем та мереж.

РН10. Здійснювати пошук інформації в різних джерелах для розв’язання задач комп’ютерної інженерії, аналізувати та оцінювати цю інформацію.

РН13. Зрозуміло і недвозначно доносити власні знання, висновки та аргументацію з питань інформаційних технологій і дотичних міжгалузевих питань до фахівців і нефахівців, зокрема до осіб, які навчаються.

РН15. Проектувати та розробляти інтелектуалізовані системи опрацювання даних для різних предметних областей.

Звіт повинен містити:

- титульний аркуш (на ньому вказують назву міністерства, назву університету, назву кафедри, номер, вид і тему роботи, виконавця та особу, що приймає звіт, рік);
- мету, варіант і завдання роботи;
- лаконічний опис теоретичних відомостей та алгоритмів вирішення задачі;
- текст програми, що обов'язково містить коментарі;
- вхідні та вихідні дані програми;
- побудовані графіки і таблиці;
- змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату А4 (210 х 297 мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші скріплюють за допомогою канцелярських скріпок. Для набору тексту звіту використовують редактор MS Word: шрифт Times New Roman, 12 пунктів. Міжрядковий інтервал: полуторний – для тексту звіту, одинарний – для листингів програм, таблиць і роздруківок даних.

ЛАБОРАТОРНА РОБОТА №1

Тема. Побудова лексичного та синтаксичного аналізаторів з використанням системи автоматизованої розробки компіляторів Coco/R

Мета. Навчитися створювати та компілювати лексичні та синтаксичні аналізатори з використанням системи автоматизованої розробки компіляторів Coco/R.

ТЕОРЕТИЧНІ ВІДОМОСТІ

1. Ознайомитися з теоретичними відомостями.

2. Якщо потрібно, інсталювати CoCo/R.

3. Ознайомитися з інтерфейсами сканера та синтаксичного аналізатора.

4. Обрати конструкції мови та описати їх в стандарті CoCo/R .

5. Згенерувати та скомпілювати лексичний та синтаксичний аналізатори.

6. Написати програму на власній мові та протестувати її виконання.

7. Зробити висновки по етапах виконання лабораторної роботи.

Список скорочень

РБНФ − розширена Бекуса-Наура форма

EBNF − extended Backus–Naur form

LL − **L**eft to right, and constructs a **L**eftmost derivation

YACC − yet another compiler compiler

Формальний опис компілятора має дві цілі:

1. Формування документа, який описує синтаксис і семантику мови.

2. Формування нотації з якої можна згенерувати ефективний компілятор.

Таким чином, системи генерації компіляторів в якій-небудь степені акцентують увагу на ці аспекти. Дана система, Coco/R, більшою мірою концентрує увагу на другій цілі, і дозволяє генерувати компілятори, що не поступаються в ефективності промисловим компіляторам, які побудовані і оптимізовані вручну. Coco/R генерує низхідні рекурсивні синтаксичні

аналізатори, які володіють простим механізмом обробки помилок і лексичним аналізатором (сканером) із спеціальною схемою буферизації вхідного потоку.

Слід відмітити, що простота і адекватність системи має не менше значення ніж ефективність системи. Як правило, програмісти не зацікавлені у використовуванні інструменту, який не зручний в роботі через наявність складної криптованої нотації та надмірної безлічі спеціальних опцій. Мається на увазі, що інструмент повинен полегшувати роботу програміста не вносячи обмеження в гнучкість використовування.

1. ОГЛЯД СИСТЕМ АВТОМАТИЗАЦІЇ РОЗРОБКИ КОМПІЛЯТОРІВ

Компілятори зараз являються важливою частиною програмного забезпечення. Це пов'язано з тим, що мови високого рівня стали основним засобом розробки програм. Крім того, постійно виникає необхідність в спеціалізованих мовах програмування для вирішення конкретних задач передачі і обробки інформації в вузьких предметних областях науки, а також компіляторах для нових архітектур комп’ютерів.

Так як розробка компіляторів є трудомісткою роботою, то в програмуванні досить швидко появився ряд різних програмних засобів для автоматизованої розробки компіляторів. Цей клас програм також називають компіляторами компіляторів. Хоча за допомогою такого інструменту можна створити як компілятор, так і інтерпретатор, транслятор, чи просто синтаксичний аналізатор. В основному процес компіляції складається з таких етапів як лексичний, синтаксичний, контекстний аналізи, проміжне представлення коду, оптимізація та генерація коду. Звичайно системи автоматизації розробки компіляторів можуть повністю автоматизувати розробку одного чи декількох етапів компіляції. Тому такі системи класифікуються за тим, які етапи вони автоматизують:

1. Генератори лексичних та синтаксичних аналізаторів. Класичним прикладом лексичного аналізатора є LEX, синтаксичного – YACC, з різноманітними їх реалізаціями. Також до таких систем відносяться AGGENT, ANAGRAM, CoCo\R, LLGEN, та ін.

2. Аналізатори потоку даних та оптимізатори. Прикладами таких систем являються BANE, OMEGA, OPTIMIX, PAG, TINY.

3. Генератори коду. До них відносяться BEG, IBURG, MBURG, TWIG.

4. Пакети програм для розробки компіляторів. Такі пакети включають програми для автоматизації декількох етапів компіляції. До них відносяться PCCTS, GENTLE, ELI, COCKTAIL.

Також до систем автоматизації розробки компіляторів відносять системи створення середовищ розробки (CENTAUR, SYNTHESIZER GENERATOR), системи атрибутної граматики(ELEGANT, RIE, OX), засоби синтаксично керованої трансляції (KIMWITU, MEMPHIS, RIGAL) та ін. Системи автоматизації розробки компіляторів належать до вузько спеціалізованих програм, тому абсолютна більшість таких програм немає зручного користувацького інтерфейсу.

Однією з найбільш простих у користуванні систем розробки компіляторів, і яку, наприклад, можна використовувати в університетських навчальних програмах, є CoCo\R. До плюсів цієї системи можна віднести докладну документацію, автоматично створюваний обробник помилок, базовими мовами для неї є всі найпопулярніші на сьогодні мови програмування (C++, Java, C#, Delphi, Ruby та багато ін.).

2. МОВА ОПИСУ КОМПІЛЯТОРА COCOL/R

2.1 Загальна структура

Опис компілятора складається з набору правил граматики, який описує лексичну і синтаксичну структуру мови, також як і трансляцію на вихідну мову. Основні елементи Cocol/R - це ідентифікатори, числа, рядки і символи:

```
ident = letter {letter | digit}.

number = digit {digit}.

string = '"' {anyButQuote} '"'.

char = '\'' anyButApostrophe '\''.
```

Букви верхнього регістра відрізняються від букв нижнього регістра. Наступні ідентифікатори - це зарезервовані ключові слова (у Java-версії Cocol/R import також зарезервоване слово):

```
ANY CONTEXT IGNORE PRAGMAS TOKENS
```

```
CHARACTERS END IGNORECASE PRODUCTIONS WEAK

COMMENTS FROM NESTED SYNC

COMPILER IF out TO
```

Коментарі обмежуються /* і */ і можуть бути вкладеними. Альтернативно вони можуть починатися з // і йти до кінця рядка. Всі описи синтаксису в Cocol/R написані в розширеній формі Бекуса-Наура (EBNF). Прийнято ідентифікатори, що починаються з символу нижнього регістра, означають термінальні позначення, ідентифікатори, що починаються з символу верхнього регістра, означають нетермінальні позначення.

Таблиця 1. Meta-символи

Символ	Пояснення	Приклад
=	розділяє сторони продукції	A = abc .
.	завершує продукцію	A = abc .
|	розділяє альтернативи	A b | c | d e
( )	групує альтернативи	(a | b) c
[ ]	позначає необов’язковий елемент	[a] b
{ }	Повторення (0 або більше разів)	{a} b

Атрибути описуютьсяі між < і >. Семантичні дії обмежуються (. і .).

Опис компілятора Cocol/R має наступну структуру:

```
Cocol =

[Imports]

“COMPILER” ident

[GlobalFieldsAndMethods]

ScannerSpecification

ParserSpecification

“END” ident ‘.’

.
```

Ім'я після ключового слова COMPILER є іменем граматики і воно має відповідати імені після ключового слова END. Ім'я граматики також означає найвище нетермінальне позначення (стартовий символ). Специфікація аналізатора повинна містити продукцію для цього позначення.

Перед ключовим словом COMPILER можна імпортувати пакети JAVA. Після граматики також можна оголосити довільні поля і методи генерованого аналізатора.

```
int sum;

void Add(int x) {

sum = sum + x;

}
```

Ці декларації написані на мові аналізатора, що генерується і не перевіряються Coco/R. Вони можуть використовуватися в семантичних діях специфікації аналізатора. Інші частини опису компілятора, що залишаються, конкретизують сканер і аналізатор, які мають генеруватися.

2.2 Специфікація сканера

Сканер повинен прочитати початковий текст, пропустити незначущі символи, розпізнати лексеми, які потім передаються синтаксичному аналізатору. Лексеми можуть бути класифіковані як літерали (ключові послідовності або класи лексем. Літерали (наприклад, “END”, “=” та ін.) записуються як стрічки і означають самих себе. Вони розміщуються в правій частині продукції, і їх не треба описувати. Узагальнені лексеми (наприклад, ідентифікатори та числа) мають певну структуру, яка повинна бути описана у вигляді регулярного виразу РБНФ. Існує багато різних видів узагальнених лексем (наприклад, множина ідентифікаторів), які розпізнаються як одна і та ж лексема.

```
ScannerSpecification =

[“IGNORECASE”]

[“CHARACTERS” {SetDecl}]

[“TOKENS” {TokenDecl}]

[“PRAGMAS” {PragmaDecl}]

{CommentDecl}
```

```
{WhiteSpaceDecl}.
```

Специфікації сканера складається з шести необов'язкових частин, які можуть розміщуватися в довільному порядку.

2.2.1 Символи

Секція CHARACTERS забезпечує опис імен для множин символів, таких як букви або цифри. Ці імена можуть тоді використовуватися в інших секціях специфікації сканера. Coco/R підтримує Unicode (UTF-8).

```
SetDecl = ident '=' Set '.'.

Set = BasicSet {('+'|'-') BasicSet}.

BasicSet = string | ident | char [".." char] | "ANY".
```

SetDecl присвоює множині символів ім'я. Базові множини символів визначаються так:

- string – множина, що складається з всіх символів стрічки;
- ident – наперед визначена множина символів з її ім'ям ;
- char – окремий символ;
- char1..char2 – множина символів від char1 до char2;
- ANY – множина всіх символі.

Множини символів можуть формуватися з базових множин за допомогою операторів:

- "+" – додавання (об'єднання) множин;
- "-" – віднімання множин.

Наприклад:

```
digit = "0123456789".

hexDigit = digit + "ABCDEF".

letter = 'A' .. 'Z'.

eol = '\r'.

noDigit = ANY - digit.
```

2.2.2 Лексеми

Секція TOKENS є головною в специфікації сканера, в якій оголошуються лексеми (або термінальні символи). Лексеми діляться на літерали і класи лексем.

Синтаксис оголошення лексем вказано нижче:

```
TokenDecl = Symbol [‘=’ TokenExpr ‘.’].

TokenExpr = TokenTerm {‘|’ TokenTerm}.

TokenTerm = TokenFactor {TokenFactor} ["CONTEXT" ‘(‘ TokenExpr
‘)’].

TokenFactor = Symbol

| ‘(‘ TokenExpr ‘)’

| ‘[‘ TokenExpr ‘]’

| ‘{‘ TokenExpr ‘}’.

Symbol = ident | string | char.
```

Лексеми можуть бути описані в довільному порядку. Оголошення лексем визначає символ з його структурою. Зазвичай символ в лівій частині опису являє собою ідентифікатор. Він вказується для того, щоб розрити суть правої частини.

Права частина специфікує структуру за допомогою регулярного виразу РБНФ. Цей вираз може містити літерали, що означають самі себе (наприклад, "END"), так як і імена множин символів (наприклад, букв), що описують довільний символ з таких множин.

Приклад:

```
ident = letter {letter | digit | '_'}.

number = digit {digit}

| "0x" hexDigit hexDigit hexDigit hexDigit.

float = digit {digit} '.' {digit} ['E' ['+'|'-'] digit {digit}].
```

2.2.3 Директиви

Директива (pragma) – це лексема, яка може зустрітися десь у вхідному потоці(це можуть бути симоли закінчення рядка або опції компілятора). Досить складно опрацьовувати всі багато численні місця, де може зустрітися

директива в граматиці. Тому існує спеціальний механізм для обробки директив без включення їх в середину продукцій. Директиви описуються подібно до лексем, але їм можна призначити семантичну дію, яка буде виконуватися не залежачи від того, розпізнані вони сканером чи ні.

```
PragmaDecl = TokenDecl [SemAction].

SemAction = "(." ArbitraryStatements ".)".
```

Приклад:

```
PRAGMAS

option = '$' {letter}. (. foreach (char ch in la.val)

if (ch == 'A') ...

else if (ch == 'B') ...

... .)
```

2.2.4 Коментарі

Коментарі досить важко (а вкладені неможливо) описати за допомогою регулярних виразів. Це викликає необхідність мати спеціальну конструкцію для обробки такої структури. Коментарі описуються специфікуванням їх відкриваючих і закриваючих обмежувачів. Можна описувати різні типи коментарів. Обмежувачі не повинні бути довшими двох символів.

```
CommentDecl = "COMMENTS" "FROM" TokenExpr "TO" TokenExpr ["NESTED"].
```

Приклад:

```
COMMENTS FROM "/*" TO "*/" NESTED

COMMENTS FROM "//" TO eol
```

Альтернативно, якщо коментарі не є вкладеними, можна описати так:

```
CHARACTERS

other = ANY - '/' - '*'.

PRAGMAS

comment = "/*" {'/' | other | '*' {'*'} other} '*' {'*'} '/'.
```

2.3 Специфікація синтаксичного аналізатора

Специфікація синтаксичного аналізатора є головною частиною опису компілятора. Вона містить продукції атрибутної граматики, що специфікують синтаксис мови, яку необхідно розпізнати, перед тим як транслювати. Продукції можуть розміщуватися в довільному порядку. Допускаються посилання на ще не описані нетермінали. Будь-яке ім’я, яке попередньо не було описано як термінальна лексема, розглядається як нетермінал.

Для кожного нетермінала має бути одна продукція (або список альтернативних правих частин). Повинна бути присутньою продукція для початкового символу, що відповідає імені граматики.

```
ParserSpecification = “PRODUCTIONS” {Production}.

Production = ident [FormalAttributes] [LocalDecl] ‘=’ Expression
‘.’.

Expression = Term {‘|’ Term}.

Term = [[Resolver] Factor {Factor}].

Factor = ["WEAK"] Symbol [ActualAttributes]

| ‘(‘ Expression ‘)’

| ‘[‘ Expression ‘]’

| ‘{‘ Expression ‘}’

| “ANY”

| “SYNC”

| SemAction.

Symbol = ident | string | char.

SemAction = “(.” ArbitraryStatements “.)”.

LocalDecl = SemAction.

FormalAttributes = ‘<’ ArbitraryText ‘>’.

ActualAttributes = ‘<’ ArbitraryText ‘>’.

Resolver = “IF” ‘(‘ {ANY} ‘)’.
```

2.3.1 Продукції

Продукції специфікують синтаксичну структуру нетерміналів. Вони мають власну область дії для атрибутів і локальних об'єктів, і складаються з лівої і правої частин, які розділені символом рівності. Ліва частина специфікує ім'я нетермінала разом з його формальними атрибутами. Права частина є контекстно-вільним виразом РБНФ, який описує структуру нетермінала, тобто спосіб його трансляції. Формальні атрибути записані як формальні параметри на мові JAVA. Вони обмежуються кутовими лапками. По аналогії з вхідними та вихідними параметрами підпрограм використовуються терміни "вхідні атрибути" і "вихідні атрибути".

2.3.2 Вирази

Вираз РБНФ описує контекстно-вільну структуру деякої частини початкової мови спільно з атрибутами і семантичними діями, які специфікують трансляцію цієї частини в необхідну мову.

```
Expression  = Term { "|" Term } .

Term  = { Factor } .

Factor = [ "WEAK" ] Symbol [ Attributes ] SemAction "ANY" "SYNC"

"(" Expression ")" "[" Expression "]" "{" Expression "}" .

Attributes  = "<" arbitraryText ">" .

SemAction  = "(." arbitraryText ".)" .

Symbol  = ident | string .
```

Нетермінали можуть мати атрибути. Вони записуються як дійсні параметри базової мови і заключаються в кутові дужки. Якщо нетермінал має формальні атрибути, кожна поява цього нетермінала повинна мати список дійсних атрибутів, який відповідає формальним атрибутам відповідно до правил сумісності мови. Відповідність, проте, перевіряється тільки тоді, коли здійснюється компіляція модуля синтаксичного аналізатора. Семантична дія - це довільна послідовність інструкцій базової мови, обмежується "(." і ".)". Символ "ANY" позначає будь-який термінал, який не є альтернативою символу "ANY". Звичайно він використовується для розбору структур, які містять довільний текст.

2.3.3 Обробка помилок

Правильне і ефективне відновлення після помилок є важкоздійсненним в синтаксичних аналізаторах, які грунтуються на ідеї низхідного рекурсивного синтаксичного аналізу, оскільки при виникненні помилки, доступно не достатньо інформації про стан процесу синтаксичного аналізу. Що необхідно зробити у разі помилки:

1. знайти всі символи, з якими синтаксичний аналіз може бути продовжений з певної позиції в доступній граматиці, починаючи з позиції помилки (відновлювані символи).

2. пропустити весь вхідний потік до появи першого символу, що входить в множину відновлюваних.

3. перемістити синтаксичний аналізатор в позицію, де відновлюваний символ може бути розпізнаний.

4. продовжити синтаксичний аналіз з цієї позиції.

У синтаксичних аналізаторах, які грунтуються на низхідному рекурсивному синтаксичному аналізі, інформація про позицію розбору і про очікувані символи неявно міститься в коді синтаксичного аналізатора, і не може бути використана для відновлення після помилок. Один з методів подолати це полягає в динамічному обчисленні відновленої множини під час розбору. Надалі, якщо виникає помилка, відновлені символи вже відомі і все, що необхідно зробити, це пропустити помилковий вхідний потік і "розкрутити" процедуру "stack up" до допустимої точки продовження.

За іншим підходом у відновленні беруть участь тільки деякі синхронізуючі точки граматики. Помилки в інших точках виявляються, але не підлягають відновленню. Розбір просто продовжується з наступної синхронізуючої точки, де граматика і вхідний потік знову відповідають один одному. Вимагається, щоб розробник граматики специфікував точки синхронізації єдиним чином. Перевага цієї ідеї полягає в тому, що ніякої множини відновлюваних символів не треба обчислювати в процесі роботи. Це робить обробник маленьким і швидким.

Програмісту необхідно враховувати декілька рекомендацій для того, щоб Coco/R генерував хороший і ефективний обробник помилок.

По-перше, необхідно специфікувати точки синхронізації. Точки синхронізації - це позиції в граматиці, де очікуються особливо надійні термінали, які важко пропустити або неправильно оформити. Коли синтаксичний аналізатор, що генерується, досягає такої точки, він підганяє вхідний потік до наступного очікуваного в цій точці символу. В багатьох мовах хорошими кандидатами на точки синхронізації є початки інструкцій (де очікується щось подібне до If, While і т.п.), початки секцій опису (де очікується щось подібне до Const, Var та ін.). Символ кінця файлу завжди один з символів синхронізації, гарантуючий, що синхронізація припиняється як мінімум в кінці вихідного тексту. Точки синхронізації специфікуються символом "SYNC".

```
Statement =
SYNC
( Designator‘=’ Expression SYNC‘;’
| “if” ‘(‘ Expression ‘)’ Statement [“else” Statement]
| “while” ‘(‘ Expression ‘)’ Statement
| ‘{‘ {Statement} ‘}’
| ...
).
```

Точка синхронізації транслюється в цикл, який пропускає всі символи, не очікувані в цій точці (виключаючи кінець файлу). Множина таких символів може бути заздалегідь обчислена на етапі генерації синтаксичного аналізатора.

2.3.4 LL(1) конфлікти

Низхідний рекурсивний синтаксичний аналіз вимагає, щоб граматика мови задовольняла властивостям LL(1). Це означає, що в будь-якій точці граматики синтаксичний аналізатор повинен бути здатний ухвалити рішення на основі єдиного символу, що йде попереду, з вибраної множини можливих альтернатив. Наприклад, наступна продукція не відноситься до LL(1):

```
Statement = ident “:=” Expression
|   ident   [ “(“ ExpressionList “)”
```

Тут, обидві альтернативи починаються з символу "ident", і синтаксичний аналізатор не може зробити вибір між продовженням йти по інструкції і пошуком "ident" як наступного вхідного символу. Проте, ця продукція може бути легко трансформована в:

```
Statement = ident (       ":=" Expression
|     [ "("ExpressionList")"    ]
```

де всі альтернативи починаються з різних символів. Існують конфлікти LL(1), які не так легко розпізнати, як в попередньому прикладі. Для програміста може бути важко знайти їх, якщо у нього немає інструменту для перевірки граматики. В результаті повинен вийти синтаксичний аналізатор, який, в деяких ситуаціях, вибирає невірну альтернативу. Coco/R перевіряє граматику на відповідність властивостям LL(1), і видає відповідні повідомлення про помилки, що допомагає скоректувати помилки.

3. ЗАСТОСУВАННЯ COCO/R

3.1 Інсталяція та використання Coco/R

Генератор Coco/R можна скачати з сторінки

```
http://www.ssw.uni-linz.ac.at/Research/Projects/Coco/
```

Там знаходиться як вже скомпільована версія Coco/R, так і комплект вихідних кодів. Для запуску програми в поточному каталозі повинні знаходитися три файли:

```
Coco.jar

Parser.frame

Scanner.frame
```

Разом з програмою поставляється досить пристойна документація: коротка і цілком читабельна. Coco/R можна запустити з командного рядка так:

```
java -jar Coco.jar fileName [Options]
```

fileName - це ім'я файлу, що містить опис компілятора. Цей файл повинен мати розширення ATG.

Атрибутивна граматика є ключовим документом для реалізації компілятора за допомогою Coco/R. При цьому користувач повинен вирішити наступні задачі:

1. написати атрибутивну граматику

2. при необхідності,написати семантичні модулі і імпортувати їх в атрибутивну граматику

3. запустити на виконання Coco/R для генерації сканера, синтаксичн ого аналізатора і головної процедури компілятора, згідно вказаної атрибутивної граматики

4. при необхідності, написати модуль, який здійснюватиме виклик головної процедури компілятора

3.2 Інтерфейс сканера

Згенерований сканер має наступний інтерфейс:

```
public class Scanner {
public Buffer buffer;
public Scanner(string sourceFile);
public Scanner(Stream s);
public Token Scan();
public Token Peek();
public void ResetPeek();
}
```

Відповідальність за відкриття файлу вхідного потоку покладена на головний клас компілятора. Метод Scan() викликається, коли потрібний наступний токен. Для недійсних токенів (викликаних неправильним синтаксисом або недійсними символами) Scan() повертає спеціальний вид лексем, які заставляють парсер вивести помилку. Peek() використовується для читання одного чи декількох токенів вперед без виключення їх з вхідного потоку.

3.3 Інтерфейс синтаксичного аналізатора

Синтаксичний аналізатор має наступний інтерфейс:

```
public class Parser {
public Scanner scanner;
public Errors errors;
public Token t;
```

```
public Token la;

public Parser(Scanner scanner);

public void Parse();

public void SemErr(string msg);

}
```

Для генерації синтаксичного аналізатора Coco/R використовує файл шаблону parser.frame. Цей файл повинен розташовуватися в поточному каталозі або в каталозі, який вказується в змінній оточення "CRFRAMES".

Coco/R автоматично перевіряє коректність граматики і інформує користувача про наявність в ній помилок. Генерація компілятора не проводиться у випадках коли:

• знайдено використовування нетермінала для якого не вказана відповідна продукція

• присутня продукція для нетермінала, який недосяжний із стартового символу

• при виявленні нетермінала, який не може бути представлений як послідовність нетерміналів

• знайдені кільцеві взаємозалежності для нетерміналів

4. РЕАЛІЗАЦІЯ ПРОЕКТУ COCO/R

4.1 Вибір конструкцій мови та опис в стандарті Coco/R

Опис мови MyLanguage для Coco\R:

```
COMPILER MyLanguage

CHARACTERS

 letter = “ABCDEFHIJKLMNOPQRSTUVWXYZabcdefhijklmnopqrstuvwxyz”.

 digit = “0123456789”.


TOKENS
```

```
 Identifier = letter {letter | digit}.
 Number = ['-'] digit {digit}.
 true = "true".
 false = "false".

COMMENTS FROM "//" TO '\n'
IGNORE '\r'+'\n'+'\t'

PRODUCTIONS
 MyLanguage = "program" Identifier "." ProgramBody.
 ProgramBody = ["var" ":" DeclareVar {DeclareVar}]
 "begin" {StatementBody} "end.".
 DeclareVar = "int" Identifier "=" Number ";"
       | "Boolean" Identifier "=" (true | false) ";".
 StatementBody = Assignment | Branching.
 Assignment =  LValue "=" (RValue [OP RValue] | "not" RValue) ";".
 OP = '+' | '-' | "and" | "or".
 LValue = Identifier.
 RValue = Identifier|Number | true | false.
 Branching = "if" CompareExpr "then" Assignment
      ["else" Assignment ].
 CompareExpr = RValue CompareOp RValue.
 CompareOp = "==" | "<>" | '<' | "<=" | '>' | ">=".
END MyLanguage.
```

4.2 Генерація і компіляція лексичного і синтаксичного аналізаторів

Згенерувати вихідний код лексичного та синтаксичного аналізаторів тепер можна за допомогою команди:

```
java -jar Coco.jar MyLanguage.atg
```

Якщо не відбудеться нічого несподіваного, на екрані (Рисунок 1) можна побачити повідомлення про успішне створення лексичного і синтаксичного аналізаторів:

Рисунок 1. Створення парсера і сканера

У поточному каталозі з'являться два файли: Scanner.java (лексичний аналізатор) і Parser.java (синтаксичний аналізатор). Перед тим ще, як транслювати в байт-код, доведеться ще вручну створити головний клас програми. В найпростішому варіанті він займе всього декілька рядків:

```
import java.* ;

public class JavaParser {

    public static void main(String[] args) {

  System.out.println("_____________________________");

      System.out.println("JavaParser (AH, May 2008)");

      if (args.length > 0) {

        System.out.println(" Reading source file " + args[0]);

        Scanner scanner = new Scanner(args[0]);

        System.out.println(" Parsing source file " + args[0]);

        Parser parser = new Parser(scanner);

        parser.Parse();

        if (parser.errors.count == 1)

          System.out.println("-- 1 error detected");
```

```
        else
          System.out.println("—" + parser.errors.count + " errors
detected");
      }
      else
        System.out.println("Syntax: JavaParser <java source file>");
    }
}
```

Передбачається, що ім'я вхідного (аналізованого) файлу передається як перший аргумент командного рядка. Програма виконує спочатку лексичний аналіз, потім синтаксичний аналіз і друкує повідомлення про кількість знайдених помилок. Якщо виявляться помилки, їх описи будуть виведені синтаксичним аналізатором автоматично.

Тепер можна запустити javac для створення виконуваних файлів:

```
javac JavaParser.java Parser.java Scanner.java
```

4.3 Тестування синтаксичного аналізатора

Якщо подати на вхід одержаному файлу JavaParser.class програму Prog.ml

```
program MyPr.
var:
int s=1;
int b=2;
begin
s=s+5;
b=b+1;
if s>b then b=1;
else s=2;
end.
```

можна одержати відповідь про її синтаксичну коректність (Рисунок 2)

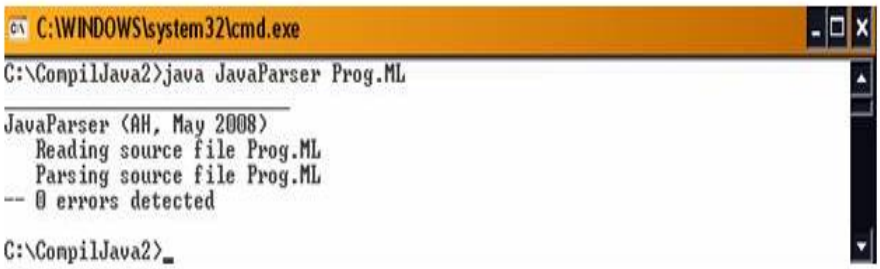

Рисунок 2. Повідомлення про синтаксичну коректність

Можна попробувати навмисно внести синтаксичні помилки, щоб переконатися в адекватній реакції парсера

```
program MyPr //пропущено крапку
var:
int s=1;
int b=2;
begn // неправильно написано begin
s=s+5;
b=b+1;
if s>b then b=1;
else s=2;
end.
```

Синтаксичний аналізатор виведе (Рисунок 3) два повідомлення про помилки

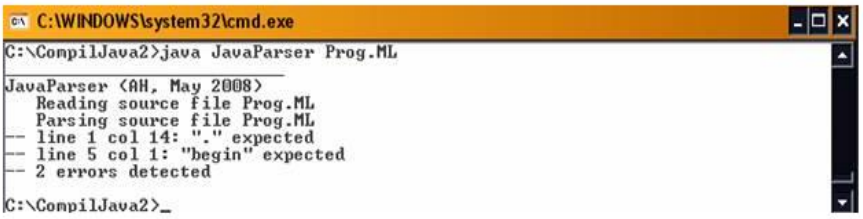

Рисунок 3. Вивід помилок

ЛАБОРАТОРНА РОБОТА №2

Тема. Оптимізація функції.

Мета. Навчитися використовувати методи оптимізації задач.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Під **оптимізацією** розуміють процес вибору найкращого варіанту з усіх можливих. Більшість задач *оптимізації* зводиться до пошуку найменшого (найбільшого) значення деякої функції. Методи математичного аналізу зручні для розв'язання цієї задачі, коли функція задається в явному вигляді і при цьому є диференційованою. Коли ж функція задається таблицею значень або має аналітично громіздку формулу, ефективними є числові методи розв'язання.

Існують різні числові *методи пошуку* для розв'язання задачі оптимізації. Вони засновані на обчисленні функції в окремих точках і виборі серед них найбільшого чи найменшого значення. Процес розв'язання задачі методом пошуку полягає у послідовному звуженні інтервалу зміни параметра функції, який називають *інтервалом невизначеності*. На початку процесу оптимізації його довжина дорівнює $b-a$, а по закінченню вона має стати меншою за допустиму похибку σ, причому $x_{n+1}-x_n<\sigma$.

Метод золотого перетину

Одним з найбільш ефективних числових методів оптимізації функції є **метод золотого перетину**. Він полягає в побудові послідовності відрізків, що стягуються до точки мінімуму (максимуму) функції. На кожному кроці, за виключенням першого, обчислення значення функції $f(x)$ проводяться лише в одній точці, яку називають *золотим перетином*.

Точка x здійснює ***золотий перетин*** відрізка $[a;b]$ якщо відношення довжини всього відрізка до довжини його більшої частини дорівнює відношенню довжини більшої частини відрізка до довжини його меншої частини (рис. 1):

$$\frac{b-a}{b-x}=\frac{b-x}{x-a}=\varphi$$

Число $\varphi=\frac{1+\sqrt{5}}{2}\approx 1.61803398874989484\ldots$ називають ***золотим числом.***

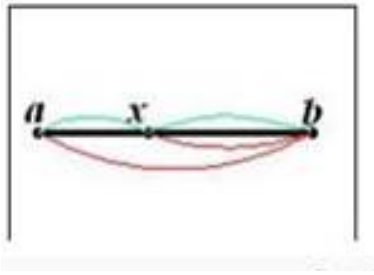

Рис. 1. Золотий перетин відрізка $[a;b]$ точкою x

Зауважимо, що на відрізку $[a;b]$ можна визначити дві симетрично розміщені відносно центру відрізка точки ($x_1 max x_2$), що реалізують золотий перетин (рис. 2). Їх знаходимо за формулами:

$$x_1 = b - \frac{b-a}{\varphi}, x_2 = a + \frac{b-a}{\varphi}$$

Якщо $a<x_1<x_2<b$, то очевидно, що точка x_1 ділить відрізок $[a;x_2]$ у відношенні золотого перетину. Аналогічно x_2 ділить відрізок $[x_2;b]$ у тій самій пропорції. Ця властивість й використовується для побудови ітераційного процесу.

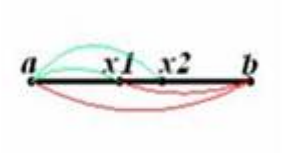

Рис. 2. Точки золотого перетину відрізка $[a;b]$

Розглянемо метод золотого перетину на прикладі знаходження мінімуму функції $f(x)$ на заданому відрізку. Припустимо, функція унімодальна, т. б. на даному відрізку вона має лише один мінімум.

На першій ітерації в середині відрізка $[a;b]$ у пропорції золотого перетину обираємо дві внутрішні точки $x_1 max x_2$ й обчислюємо значення функції у цих точках. Якщо $f(x_1) < f(x_2)$, очевидно, що мінімум функції розташований на одному з відрізків: $[a;x_1]$ чи $[x_1;x_2]$.Тому відрізок $[x_2;b]$ можна відкинути, зменшивши тим самим початковий інтервал невизначеності. Другу ітерацію проводимо на новому відрізку $[a;b]$, ввівши позначення:

$$b = x_2, x_2 = x_1, x_2 = a + \frac{b-a}{\varphi}$$

Якщо ж $f(x_1) > f(x_2)$, очевидно, що мінімум функції розташований на одному з відрізків: $[x_1;x_2]$ чи $[x_2;b]$. Отже можна відкинути відрізок $[a;x_1]$.

Другу ітерацію в цьому випадку проводимо на новому відрізку $[a;b]$ ввівши позначення:

$$a = x_1, x_1 = x_2, x_1 = b - \frac{b-a}{\varphi}$$

Знову обчислюємо значення функції $f(x_1)$ і $f(x_2)$, проводимо порівняння та повторюємо алгоритм звуження інтервалу невизначеності.

Процес оптимізації триває до тих пір, поки довжина чергового відрізка $[a;b]$ не стане меншою за задану величину σ $|b^{(k)} - a^{(k)}| < \sigma, де k = 0,1,2,3\ldots$

Завдання

Методом золотого перетину знайти найменше (найбільше) значення унімодальної функції ***f(x)*** на заданому відрізку $[a;b]$, з точністю до 0.01.

Вар.	*Функція*	*[a,b]*
1	$f(x)=-x^2-e^{-0.85x}$	[–1;3]
2	$f(x)=x^4-1.3arctg(1.5x)$	[–2;2]
3	$f(x)=4x-e^{(x-0.2)}$	[–1;3]
4	$f(x)=x^2+2e^{-0.85x}$	[–1;3]
5	$f(x)=1.5arctg(x)-x^4$	[–2;2]
6	$f(x)=e^{(x-0.4)}-3.4x$	[–1;3]
7	$f(x)=-x^2-3e^{-0.45x}$	[–2;3]
8	$f(x)=x^4-0.9arctg(2.5x)$	[–2;2]
9	$f(x)=2.8x-e^{(x-0.6)}$	[–1;3]
10	$f(x)=x^2+4e^{-0.25x}$	[–3;2]
11	$f(x)=1.1arctg(2x)-x^4$	[–2;2]
12	$f(x)=e^{(x-0.8)}-2.2x$	[–1;4]
13	$f(x)=-x^2-5e^{-0.05x}$	[–2;3]
14	$f(x)=x^4-0.3arctg(3.5x)$	[–2;2]
15	$f(x)=1.6x-e^{(x-1)}$	[–1;3]
16	$f(x)=x^2+6e^{0.15x}$	[–2;3]
17	$f(x)=0.7arctg(3x)-x^4$	[–2;2]
18	$f(x)=e^{(x-1.2)}-x$	[–1;3]
19	$f(x)=-x^2-7e^{0.35x}$	[–4;1]
20	$f(x)=x^4+0.2arctg(4.5x)$	[–2;2]

Зразок виконання завдання

Методом золотого перетину знайти мінімальне значення унімодальної функції $f(x)=x^2+10e^{0.95x}$ на відрізку $[-5;5]$, з точністю до 0.01.

Розв'язання

Обчислення проводимо за формулами:

$$x_1=b-\frac{b-a}{\varphi},\ x_2=a+\frac{b-a}{\varphi}$$

Результати обчислень заносимо до таблиці:

k	$a^{(k)}$	$b^{(k)}$	$x_1^{(k)}$	$x_2^{(k)}$	$f(x_1^{(k)})$	$f(x_2^{(k)})$
1	-5	5	-1.18047	1.18047	4.65159	32.08641
2	-5	1.18047	-2.63935	-1.18047	7.78098	4.65159
3	-2.63935	1.18047	-1.18047	-0.27852	4.65159	7.75273
4	-2.63935	-0.27852	-1.73763	-1.18047	4.93841	4.65159
5	-1.73763	-0.27852	-1.18047	-0.83583	4.65159	5.21877
6	-1.73763	-0.83580	-1.39317	-1.18050	4.60291	4.65157
7	-1.73763	-1.18050	-1.52483	-1.39320	4.67412	4.60292
8	-1.52480	-1.18050	-1.39320	-1.31201	4.60292	4.59672
9	-1.39320	-1.18050	-1.31200	-1.26174	4.59672	4.60799
10	-1.39320	-1.26170	-1.34297	-1.31200	4.59558	4.59672
11	-1.39320	-1.31200	-1.36219	-1.34300	4.59706	4.59558
12	-1.36220	-1.31200	-1.34300	-1.33117	4.59558	4.59550
13	-1.34300	-1.31200	-1.33120	-1.32384	4.59550	4.59577
14	-1.34300	-1.32380	-1.33567	-1.33120	4.59546	4.59550
15	-1.34300	-1.33120	-1.33849	-1.33567	4.59548	4.59546
16	-1.33849	-1.33120	-1.33567	-1.33398	4.59546	4.59547

Критерієм закінчення ітераційного процесу є виконання умови

$$|b^{(k)} - a^{(k)}| < \sigma, \partial e k = 0, 1, 2, 3 \ldots.$$

Оскільки:

$$|b^{(16)} - a^{(16)}| = |-133120 + 1.33849| = 0.00729 < 0.01,$$

то ітераційний процес зупиняється. Можемо прийняти

$$x_{min} \approx \frac{1}{2}(-1.33849 - 1.33120) \approx -1.33485 \approx -1.33$$

Отже

$$y_{min} = f(x_{min}) = (-1.33)^2 + 10e^{0.95(-1.95)} \approx 4.59553 \approx 4.60$$

Відповідь: $x_{min} \approx -1.33, y_{min} \approx 4.60$

Звіт про виконання лабораторної роботи повинен містити:

- Назва, мета та завдання лабораторної роботи
- Зміст індивідуального завдання
- Лістинг програми або послідовність розв'язку задачі
- Скріншоти виконання програм, результати
- Висновки

ЛАБОРАТОРНА РОБОТА №3

Тема. Онтологічне моделювання предметної області в програмі Protégé

Мета. Навчитися моделювати онтологію предметної області в Protégé. Встановлення та налаштування програмних засобів.

ТЕОРЕТИЧНІ ВІДОМОСТІ

1. Онтологічний підхід в системах електронного навчання.

Онтології використовуються в штучному інтелекті, семантичній мережі, системній інженерії, програмній інженерії, біомедичній інформатиці, бібліотечній науці та інформаційній архітектурі у формі подання знань про світ або його частини.

Існують чотири категорії онтології: статична, динамічна, навмисна і соціальна. Статична онтологія описує речі, які існують, їх атрибути і відносини. Динамічна онтологія описує світ в термінах станів, переходів і процесів. Навмисна онтологія охоплює світ агентів, речі, у які вірять, хочуть, доводять або спростовують і про які сперечаються. Соціальна онтологія охоплює соціальні явища, постійні організаційні структури або змінні мережі спілок і незалежних організацій.

Попередній набір критеріїв проектування для онтологій, метою яких є обмін знаннями та взаємодія між програмами на основі загальної концепції приведено в таблиці 1.

Таблиця 1

Набір критеріїв проектування для онтологій

№	Критерій	Опис
1	Ясність	Онтологія повинна повідомляти запропонований сенс визначених термінів. Визначення повинні бути об'єктивними і незалежними від соціального і обчислювального контексту. Повне визначення (предикат, який визначається необхідними і достатніми умовами) є кращим від часткового визначення (визначається тільки необхідними або достатніми умовами). Всі визначення повинні бути задокументовані з використанням природної мови
2	Узгодженість	Онтологія повинна дозволяти висновки, які узгоджуються з визначеннями. Визначені аксіоми повинні бути логічно послідовними. Узгодженість також повинна застосовуватися до понять, які визначені неформально, наприклад, як описано в документації на природній мові і прикладах. Якщо пропозиція, яка може бути виведена з аксіом, суперечить визначенню чи прикладу неформально, то онтологія є некогерентною
3	Розширюваність	Онтологія повинна бути спроектована так, щоб передбачити використання спільного словника. Він повинен запропонувати концептуальну основу для цілого ряду передбачуваних завдань, і

		представлення повинно бути створено таким чином, щоб можна було монотонно розширювати і спеціалізувати онтологію. Іншими словами, потрібно мати можливість визначати нові терміни, для спеціальних видів використання, на основі існуючого словника, не вимагаючи, при цьому, перегляду існуючих визначень
4	Мінімальна упередженість кодування	Концептуалізація повинна бути вказана на рівні знань без конкретного кодування на рівні символу. Упередженість кодування виникає, коли вибір представлення робиться виключно для зручності позначення або реалізації. Упередженість кодування потрібно звести до мінімуму, оскільки агенти обміну знаннями можуть бути реалізовані в різних системах і стилях представлення
5	Мінімальні онтологічні зобов'язання	Онтологія повинна вимагати мінімального онтологічного зобов'язання, достатнього для підтримки передбачуваного обміну знаннями. Онтологія повинна робити якомога менше тверджень про те, як моделюється світ, дозволяючи розробникам, спеціалізувати та вдосконалювати онтологію в міру необхідності

Кожна інформаційна система має свою власну онтологію. При дослідження впливу онтології на інформаційну систему, розрізняють два ортогональних виміри: часовий – стосується використання онтології під час розробки або при запуску і структурний вимір – конкретний спосіб, який може використовувати онтологія для впливу на основні компоненти інформаційної системи.

Якщо онтологія використовується інформаційною системою під час виконання, розуміється інформаційна система, керована онтологією. Під час розробки – розробка інформаційної системи, заснованої на онтології.

В контексті часу виконання розрізняють два різні сценарії. У першому сценарії існує певний набір повторно використовуваних онтологій, організований у бібліотеці онтологій, що містить домени та онтології завдань. У другому сценарії повторне використання обмежено, так як є лише загальна онтологія, що складається з різниці між різними рівнями домену серед основних сутностей у світі та відмінностей в мета-рівнях відносно видів класу та видів відносин.

В контексті часу розробки потрібно відрізняти інформаційну систему, орієнтовану на онтологію, від інформаційної системи, що заснована на онтології: в першому випадку компонент інформаційної системи підтверджує існування онтології і може використовувати її, при необхідності, для будь-якого конкретного застосування. У другому випадку онтологія це компонент, який співпрацює, під час виконання, в напрямку "вищої" спільної мети інформаційної системи.

Важливим аспектом використання онтології під час виконання є можливість взаємодії між програмними агентами. Програмні агенти взаємодіють один з одним через повідомлення, що містять висловлювання, сформульовані в термінах онтології (керований онтологічний зв'язок). Щоб

програмний агент міг зрозуміти сенс цих виразів, він потребує доступу до цієї онтології.

Програмне забезпечення є важливою частиною багатьох інформаційних систем і, зазвичай, містить знання про домени, які не зберігаються явно в базі даних. Під час розробки, розробник інформаційної системи може створювати статичну частину програми за допомогою онтології. Під час виконання можна явно вказати всі знання домену, неявно закодовані в прикладній програмі, перетворюючи програму в систему, засновану на знаннях. Як відомо, це має великі переваги з точки зору простоти обслуговування, розширюваності та гнучкості. У цьому випадку база знань може складатися з базової бази знань та онтології. Онтології можуть допомогти збільшити прозорість прикладного програмного забезпечення.

Іншим підходом до створення інтелектуальних веб-програм є написання програмного коду на мові загального призначення (наприклад, C, Perl, Java, Lisp, Python або XSLT), це дозволить оновлювати дані з різних місць. Код для цієї мети часто організовується в реляційній базі даних у вигляді збережених процедур, в XML-додатках використовуються мови перетворення, такі як XSLT.

Семантична модель – один з видів моделі знань. Семантична модель складається з мережі понять і відносин між цими поняттями. Концепція це ідея або тема, якою зацікавлений користувач. Концепцію і відносини називають онтологією – семантична модель, що описує знання. Семантична модель дозволяє користувачам задавати питання природним чином і допомагає виявляти закономірності і тенденції в інформації та зв’язки між її розрізненими фрагментами.

Стандарти семантичної павутини використовують ідею ієрархії класів, що представляють спільність і мінливість. Оскільки, семантична павутина, на відміну від ООП, не фокусується на представленні програмного забезпечення, то класи не визначаються з точки зору поведінки функцій. Але поняття класів та підкласів залишилося і вони виконують ті ж задачі. Класи високого рівня представляють собою сукупність великої різноманітності сутностей, тоді як класи нижнього рівня являють собою невеликий, конкретний набір елементів.

Модель може надати структуру (наприклад, класи і підкласи) для представлення та організації спільності і мінливості точок зору, коли вони відомі. Однак, в розширеній версії такої організації, модель може забезпечити структуру для опису того, що можна сказати про певний елемент. Існує певна різниця між даними в Інтернеті. Ця різниця є одним з рівнів виразності. Різні структури виразного моделювання це різні інструменти для різних цілей.

Semantic Web надає ряд мов моделювання, які відрізняються за рівнем виразності, тобто вони представляють собою різні інструменти, які дозволяють різним людям представляти різні види інформації. У таблиці 2 наведено мови семантичної павутини від найменш виразних до найвиразніших.

Таблиця 2

Мови семантичної павутини

№	Назва мови	Опис мови
1.	RDF	Це базовий фреймворк, на якому побудовано решту семантичної павутини. RDF надає механізм, що дозволяє зробити базові вислови про певну область в одній моделі. RDF є рекомендацією W3C з 2003 року
2.	RDFS	RDFS – мова з виразністю для опису основних понять спільності і мінливості, подібна до об'єктних мов та інших систем класів, тому, що використовує класи, підкласи і властивості. RDFS є рекомендацією W3C з 2003 року
3.	RDFS-Plus	RDFS-Plus це підмножина OWL, більш виразна, ніж RDFS, але без складності OWL. RDFS-Plus включає в себе достатню виразність для опису того, як можна використовувати певні властивості і залежності. RDFS-Plus не стандартизовано
4.	OWL	OWL забезпечує виразність логіки в семантичній мережі. Це дозволяє встановлювати обмеження між класами, об'єктами і властивостями. OWL була прийнята в якості рекомендації W3C у 2003 році

Семантична павутина, як і документована, заснована на деяких радикальних поняттях обміну інформацією. Ці ідеї забезпечують середовище, для розвитку процесу обміну, і можливий мережевий ефект взаємодії знань. Але такий процес збору інформації створює плутанину, розбіжності й конфлікти. Для вирішення цих проблем використовується моделювання.

Моделювання – процес організації інформації для використання певною групою людей. Моделювання підтримує цей процес трьома способами:

– надає фреймворк для комунікації;
– надає засоби для пояснення висновків;
– забезпечує структуру для управління різними точками зору.

Онтологія моделей циклічних сигналів подана в середовищі Protégé.

2. Мова опису онтологій OWL

OWL – мова онтології та один з основних стандартів семантичної павутини. OWL використовується в двох основних напрямах:

– швидке та гнучке моделювання даних;
– ефективне автоматичне обґрунтування.

Існує чотири основні групи мов програмування:

– імперативні мови;
– мови запитів;
– мови даних;
– мови моделювання.

OWL є мовою моделювання. Вона має ряд переваг у порівнянні із застарілими мовами (XSD, UML і SQL), яких достатньо для побудови класів та

властивостей і створення простих ієрархічних відносин. SQL дозволяє створювати нову таблицю для кожного класу, додавати новий стовпець для кожної властивості і задавати базові відносини з використанням зовнішніх ключів. Проте, SQL не дозволяє представляти відносини підкласів. Більш виразні мови, такі як UML, роблять статичні підкласи, але не можуть представляти динамічні відносини.

Однією з відмінних особливостей OWL є те, що вона може бути використана для вираження складних уявлень про дані. В даний час більшість технологій, які використовують мови моделювання даних, розроблені з використанням жорсткого підходу. Такий підхід створює проблему зміни властивостей в реляційній базі даних. Якщо властивість визначено однозначно, але її потрібно зробити багатозначною, то для багатьох сучасних реляційних баз даних така зміна вимагатиме видалення всього стовпця для цієї властивості, а потім створення нової таблиці, яка міститиме всі значення властивостей, а також посилання на зовнішній ключ. Як наслідок, це приводить до втрати часу та індексів оригінальної таблиці. Це також призведе до втрати будь-яких пов'язаних користувацьких запитів. Тому така зміна складна в реалізації, а інколи практично неможлива.

Всі операції моделювання даних у OWL є RDF-трійками, тому є поступовими. Покращення або модифікація моделі даних може бути здійснено шляхом зміни відповідної RDF-трійки. OWL дозволяє використовувати модель даних для підтримки різних задач міркування. Здатність підтримувати аргументацію даних дозволяє розробникам мінімізувати дані, які явно зберігаються, і складність запитів, необхідних для отримання цих даних. Однак, на сучасних комп’ютерах, деякі види міркувань можна виконувати набагато швидше, ніж інші. Тому, OWL має ряд вбудованих профілів, які дозволяють розробникам користуватися тими аргументами, які задовольняють відповідні цілі продуктивності.

Для створення онтологій з використанням OWL доступно багато програмного забезпечення – Protégé, TopBraid Composer та ін., також можна використати будь-який текстовий редактор. На відміну від деяких інших мов, які завжди представлено в одному вигляді, OWL можна записати різними способами. OWL може бути виражено в канонічному форматі OWL/XML, а також в форматі RDF/XML.

Виразність, гнучкість і ефективність OWL роблять її ідеальною мовою моделювання для створення веб-онтологій, які представляють виключно складні представлення даних. Існують різні варіанти OWL: OWL 2/Full, OWL 2/EL, OWL 2/QL, OWL 2/RL. Кожен з яких є окремою підмножиною повного стандарту OWL. Ці варіанти є простішими і, в деяких випадках, набагато ефективнішими, ніж повний стандарт. Вибір певного варіанту мови OWL залежить від особливостей додатка:

- якщо потрібні розумні міркування про складні класи і властивості, слід використовувати OWL 2/EL;

- якщо є великий обсяг даних екземпляра, і потрібно ефективно робити запити до них, використовуючи деякі помірно складні відносини класів, доречно застосувати OWL 2/QL;

- у випадку більш складних відносин між класами, що ґрунтуються на типовому рушієві бізнес-правил, підійде варіант OWL 2/RL.

Однак, для моделювання лише власних даних без аргументатора, слід використовувати OWL 2/Full

2. Загальний процес розробки онтологій в середовищі Protégé. Для побудови таксономії моделей циклічних сигналів використано середовище Protégé (рис. 1), це безкоштовна платформа з відкритим вихідним кодом, яка надає набір інструментів для створення моделей доменів і додатків на основі знань з онтологіями.

Після встановлення та запуску середовища Protégé відкриється головне вікно програми. Програма автоматично повідомить про наявність нових оновлень і надасть можливість вибрати, які елементи слід оновити.

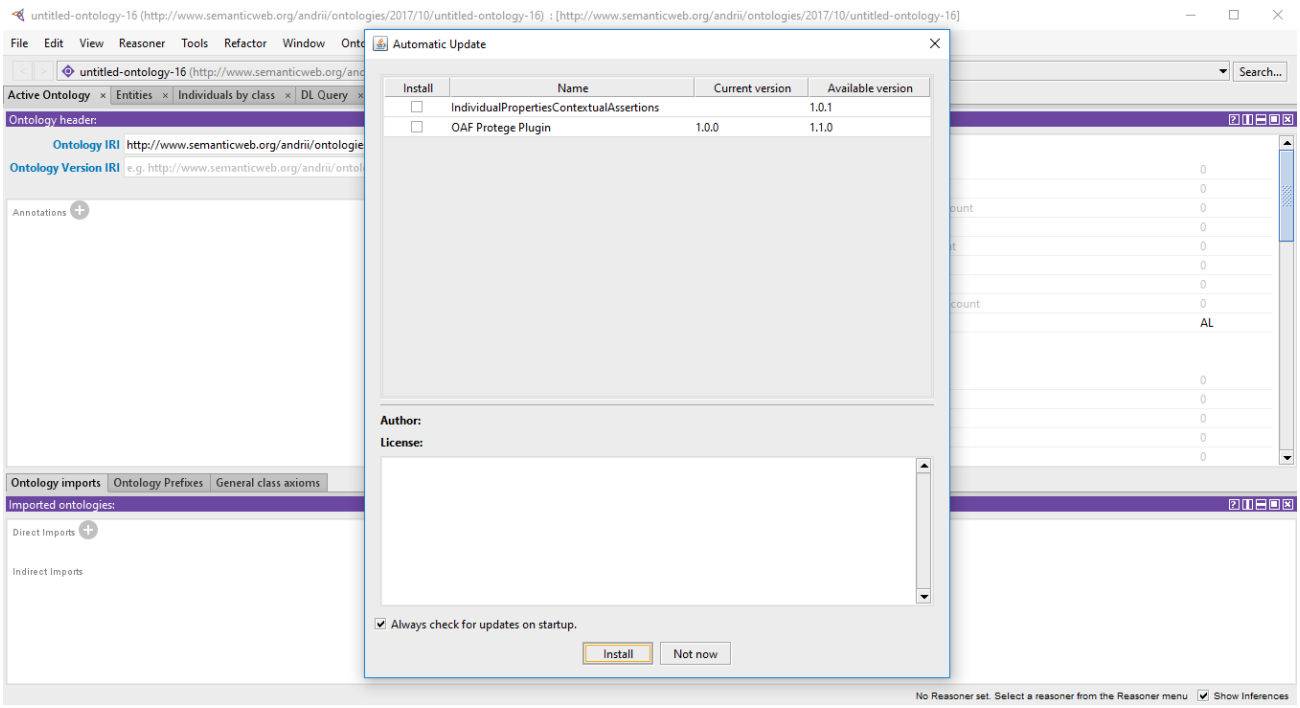

Рис. 1. Головне вікно середовища Protégé

Також, автоматично відкриється порожній проект, його потрібно зберегти вибравши в меню “File” і “Save”. Після цього відкриється вікно, в якому потрібно вибрати формат онтології, як це показано на рис. 2.

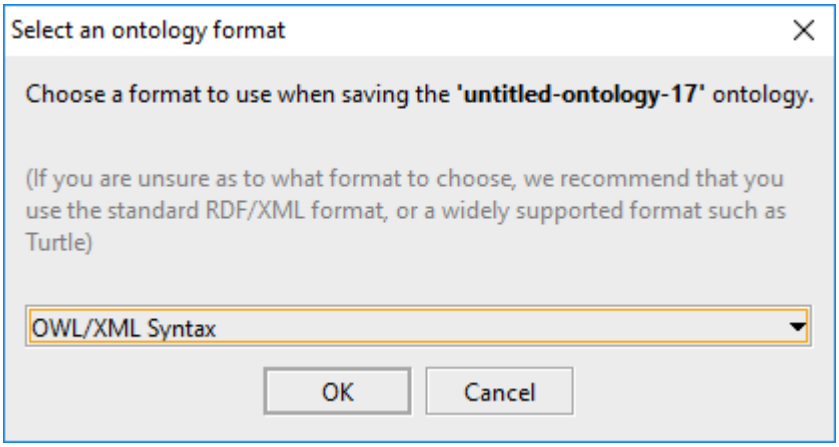

Рис. 2. Вікно вибору формату онтології

У наступному вікні, показаному на рис. 3, потрібно вказати назву проекту та натиснути кнопку “Save”.

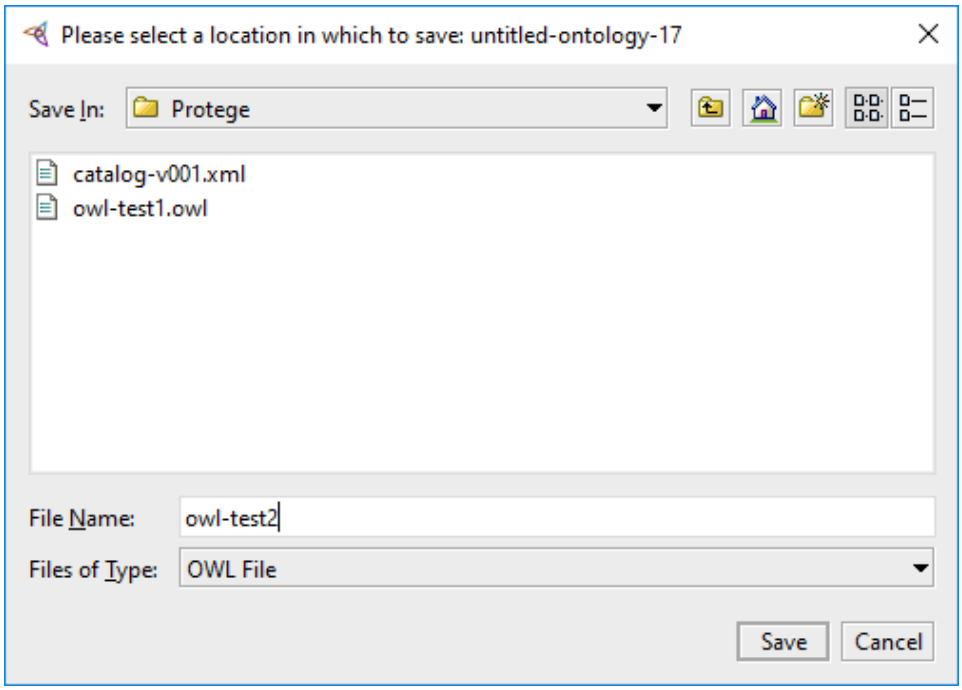

Рис. 3. Вікно збереження проекту

Далі потрібно перейти на вкладку “Entities” та вибрати “Classes” (рис. 4). У новоствореній онтології є лише клас owl:Thing, він відображає множину всіх об’єктів, оскільки всі наступні класи є його підкласами.

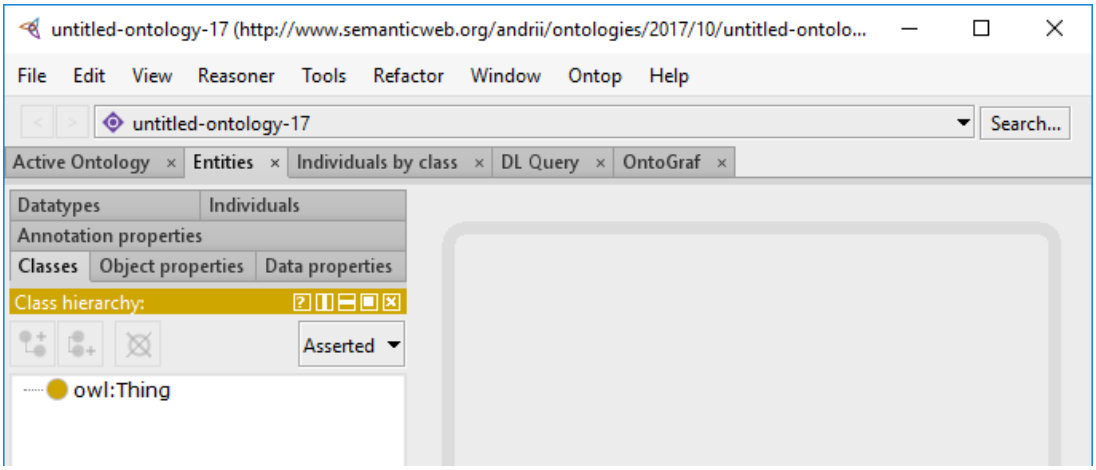

Рис. 4. Вікно вкладки «Classes»

Для створення нових класів та підкласів потрібно викликати контекстне меню класу owl:Thing, за допомогою правої кнопки миші, та вибрати “Add Subclass…” або “Add Sibling Class…” відповідно.

Графічне представлення класів та підкласів у вигляді дерева (рис. 5) здійснюється за допомогою вбудованого додатку – OntoGraf.

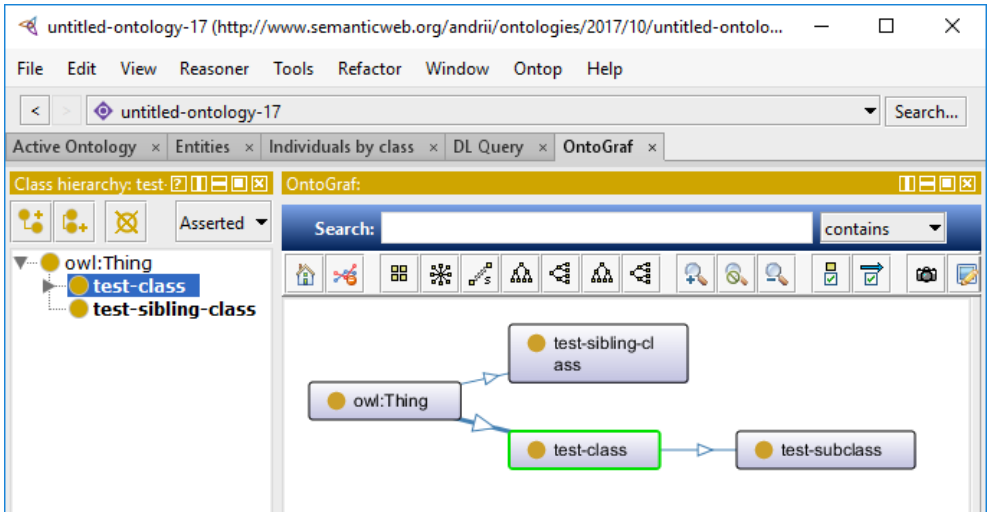

Рис. 5. Представлення класів та підкласів у середовищі Protégé

Додаток OntoGraf вимкнено позамовчуванню, для того, щоб його використовувати потрібно у пункті “Tabs” меню “Window” вибрати OntoGraf.

Отже, шляхом додавання нових класів та підкласів будується таксономія предметної області.

3. Онтологія моделей циклічних сигналів. Одним із завдань лабораторної роботи є створення онтології електронного підручника “Теоретичні основи моделювання та опрацювання циклічних сигналів в інформаційних системах”.

Для виконання цієї задачі було розроблено:

- таксономічне дерево класів циклічних функцій теорії циклічних функціональних відношень, шляхом певної генеративної процедури;

- метод параметричного генерування вузлів та дуг таксономічного класифікаційного дерева моделей циклічних сигналів в рамках теорії циклічних функціональних відношень.

При створення методу потрібно розглянути означення циклічного функціонального відношення як тотожно істинний чотирьохмісний предикат, а саме, як функцію-висловлювання $P(x_1, x_2, x_3, x_4)$, яку задано на множинах X_Ψ, X_A, $X_{T(t,n)}$, X_W, ($x_1 \in X_\Psi$, $x_2 \in X_A$, $x_3 \in X_{T(t,n)}$, $x_4 \in X_W$), і яка набирає своїх значень із множини Def_{cf} всіх можливих означень конкретних підкласів циклічних функціональних відношень. Множини X_Ψ, X_A, $X_{T(t,n)}$, X_W репрезентують наперед виділені (окреслені, задані) сукупності (класи) відповідних математичних об'єктів, а саме, можливі області значень, атрибути, типи функцій ритму, можливі області визначення. Із класами X_Ψ, X_A, $X_{T(t,n)}$, X_W можна пов'язати відповідні їм таксономії T_Ψ, T_A, $T_{T(t,n)}$, T_W, які є реляційними системами.

Метод параметричного генерування вузлів та дуг таксономічного класифікаційного дерева моделей циклічних сигналів, в рамках теорії циклічних функціональних відношень, полягає в реалізації таких базових кроків:

- формування області визначення предиката $P(x_1, x_2, x_3, x_4)$, а саме окреслення множин X_Ψ, X_A, $X_{T(t,n)}$, X_W та їх назв;
- формування таксономій T_Ψ, T_A, $T_{T(t,n)}$, T_W у вигляді кодованих (числове і словесне кодування вузлів) таксономічних дерев із множин X_Ψ, X_A, $X_{T(t,n)}$, X_W;
- формування таксономії термінів (назв) класів моделей циклічних сигналів згідно побудованої генеративної граматики;
- формування глосарію та таксономії класів циклічних функціональних відношень із базового означення абстрактної циклічної функції та таксономій T_Ψ, T_A, $T_{T(t,n)}$, T_W на основі генеративної процедури.

Першим етапом реалізації процедури параметричного генерування вузлів та дуг таксономічного класифікаційного дерева моделей циклічних сигналів є формування множин X_Ψ, X_A, $X_{T(t,n)}$, X_W, на яких задано предикат $P(x_1, x_2, x_3, x_4)$, та назви елементів.

Елементами множини X_Ψ є можливі (допустимі) області значень Ψ абстрактної циклічної функції, які загалом є деякими лінійними просторами над полем дійсних або комплексних чисел. У залежності від підходу (детермінованого, стохастичного, нечіткого, інтервального) до врахування невизначеності структури циклічних сигналів, множину X_Ψ можна розбити на чотири підмножини $X_{\Psi d}$, $X_{\Psi s}$, $X_{\Psi f}$, $X_{\Psi i}$, ($X_\Psi = X_{\Psi d} \cup X_{\Psi s} \cup X_{\Psi f} \cup X_{\Psi i}$), які не мають спільних елементів, як це показано на рис. 6.

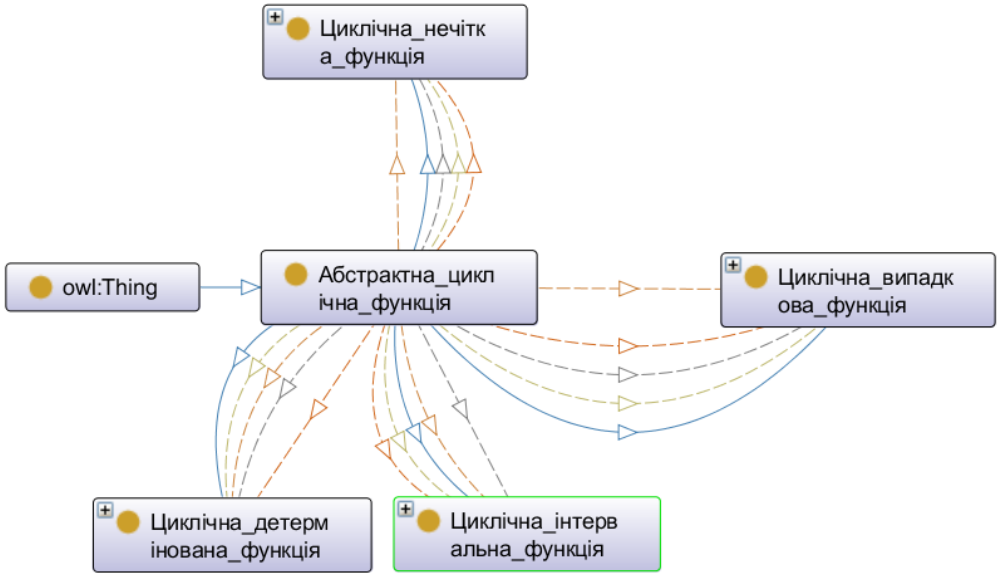

Рис. 6. Абстрактна циклічна функція

Множина $X_{\Psi d}$ – множина можливих лінійних просторів, які використовуються як області значень циклічних функцій при детермінованому підході до моделюванні сигналів. Клас циклічних функцій із такими областями значень названо циклічними детермінованими функціями.

Множина $X_{\Psi s}$ – множина можливих лінійних просторів, які використовуються як області значень циклічних функцій при стохастичному (ймовірнісному, випадковому) підході до моделюванні сигналів. Клас циклічних функцій із такими областями значень названо циклічними випадковими функціями.

Множина $X_{\Psi f}$ є множиною можливих лінійних просторів, які використовуються як області значень циклічних функцій при нечіткому підході до моделюванні сигналів. Клас циклічних функцій із такими областями значень названо циклічними нечіткими функціями.

Множина $X_{\Psi i}$ це множина можливих лінійних просторів, які використовуються як області значень циклічних функцій при інтервальному підході до моделювання сигналів. Клас циклічних функцій із такими областями значень названо циклічними інтервальним функціями.

Елементами $X_{\Psi d}$ є наступні лінійні простора:

- лінійний простір дійсних ($\Psi = R$) та комплексних чисел ($\Psi = C$);
- лінійний простір векторів над полем дійсних ($\Psi = R^n$) та комплексних чисел ($\Psi = C^n$);
- лінійний простір матриць над полем дійсних ($\Psi = M_{nm}(R)$) та комплексних чисел ($\Psi = M_{nm}(C)$);
- лінійний простір тензорів ($\Psi = T_{nm}(R)$);
- лінійний простір функцій;

– лінійний простір операторів.

Базуючись на відповідних класах областей значень, можна побудувати структуру циклічних детермінованих функцій, як це показано на рис. 7.

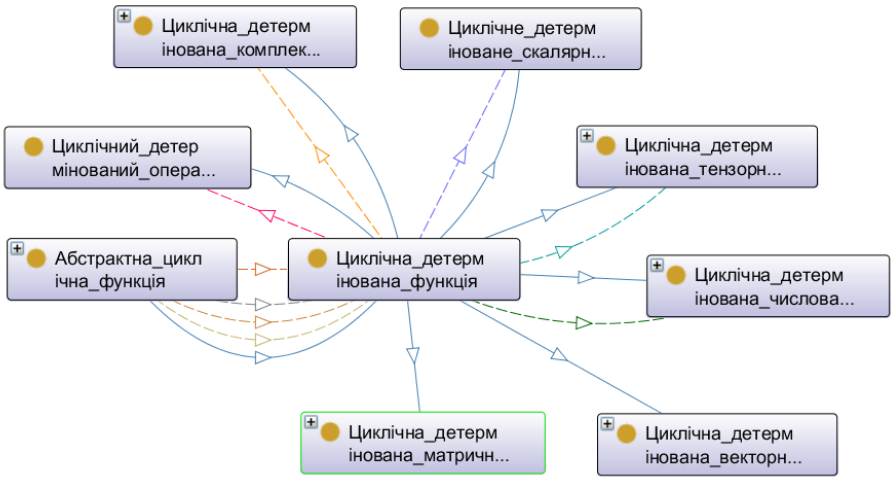

Рис. 7. Циклічні детерміновані функції

Елементами $X\Psi_s$ є такі лінійні простори:

– лінійний простір випадкових величин, заданих на одному і тому ж ймовірнісному просторі ($\Psi = L2(\Omega, P)$);

– лінійний простір випадкових векторів $\Psi = L_n2(\Omega, P)$;

– лінійний простір випадкових матриць ($\Psi = M_{nm}(L2(\Omega, P))$;

– лінійний простір випадкових тензорів $\Psi = T_{nm}(L2(\Omega, P))$;

– лінійний простір випадкових функцій;

– лінійний простір випадкових операторів.

Базуючись на відповідних класах областей значень, можна побудувати структуру циклічних випадкових функцій, як це показано на рис. 8.

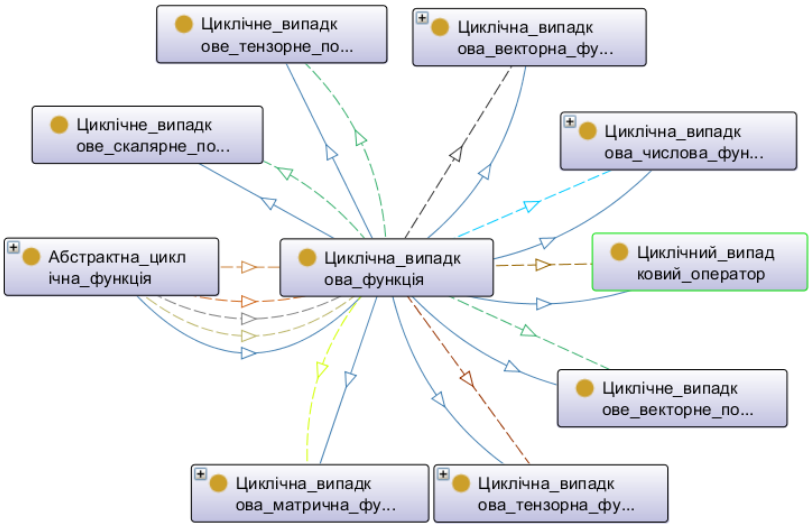

Рис. 8. Циклічні випадкові функції

Лінійні простори множини $X\Psi_f$:

– простір нечітких чисел;

– лінійний простір нечітких векторів;
– лінійний простір нечітких матриць;
– лінійний простір нечітких тензорів;
– лінійний простір нечітких функцій;
– лінійний простір нечітких операторів.

Відповідно до класів області значень, можна побудувати нечіткі функції, як це показано на рис. 9.

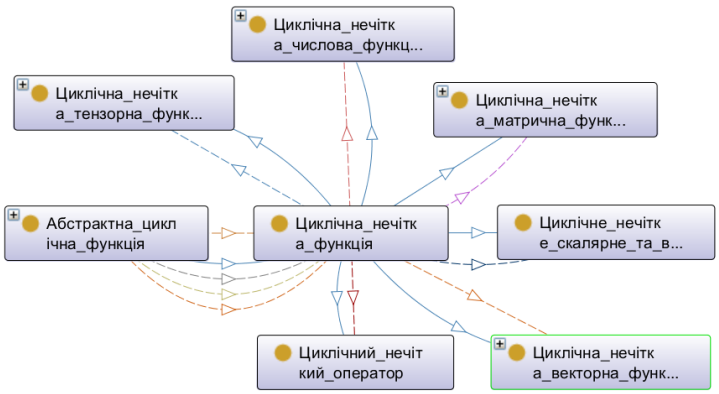

Рис. 9. Циклічні нечіткі функції

Прикладами елементів $X_{\Psi i}$ можуть бути такі лінійні простори:
– лінійний простір числових інтервалів;
– лінійний простір векторів числових інтервалів;
– лінійний простір матриць числових інтервалів;
– лінійний простір тензорів числових інтервалів;
– лінійний простір функцій числових інтервалів;
– лінійний простір операторів числових інтервалів.

Базуючись на відповідних класах областей значень, можна навести приклади циклічних нечітких функцій (рис. 10).

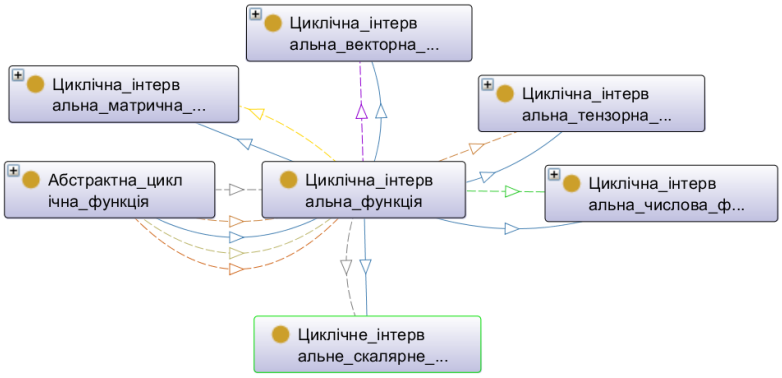

Рис. 10. Циклічні інтервальні функції

Елементами множини X_A є відображення $\{p_k : \Psi^{n_k} \to A_k, k = \overline{1,K}\}$, якщо розглядається багатовимірна циклічна структура або p: $\Psi \to A$, якщо розглядається одновимірна циклічна структура. Елементи множини X_A залежать від елементів множини X_Ψ і вибираються із практичних міркувань конкретних задач моделювання та опрацювання сигналів.

Елементами множини $X_{T(t,n)}$ є різні типи (класи) функцій ритму $T(t,n)$ циклічного функціонального відношення. Зокрема такими елементами множини $X_{T(t,n)}$ є класи стабільного ($T(t,n) = n * T$) та змінного ритму ($T(t,n) \neq n * T$).

Елементами множини X_W є допустимі області визначення W циклічної функції:

-$W = R^n$, де $n \in N$, тобто областю визначення циклічного функціонального відношення є n-вимірний Евклідовий простір над полем дійсних чисел, то будуть циклічні поля неперервного аргументу;

-$W = D^n$, тобто областю визначення циклічного функціонального відношення є n-вимірний Евклідовий простір над зліченною підмножиною ізольованих точок із поля дійсних чисел, то будуть циклічні поля дискретного аргументу;

-W = R, то буде отримано циклічну функцію неперервного аргументу;

-W = D, то буде отримано циклічну функцію дискретного аргументу;

-W = D = Z є циклічною послідовністю, яка може мати лише постійний ритм, тобто бути періодичною.

Детальну онтологію представлено на наступному рис. 11.

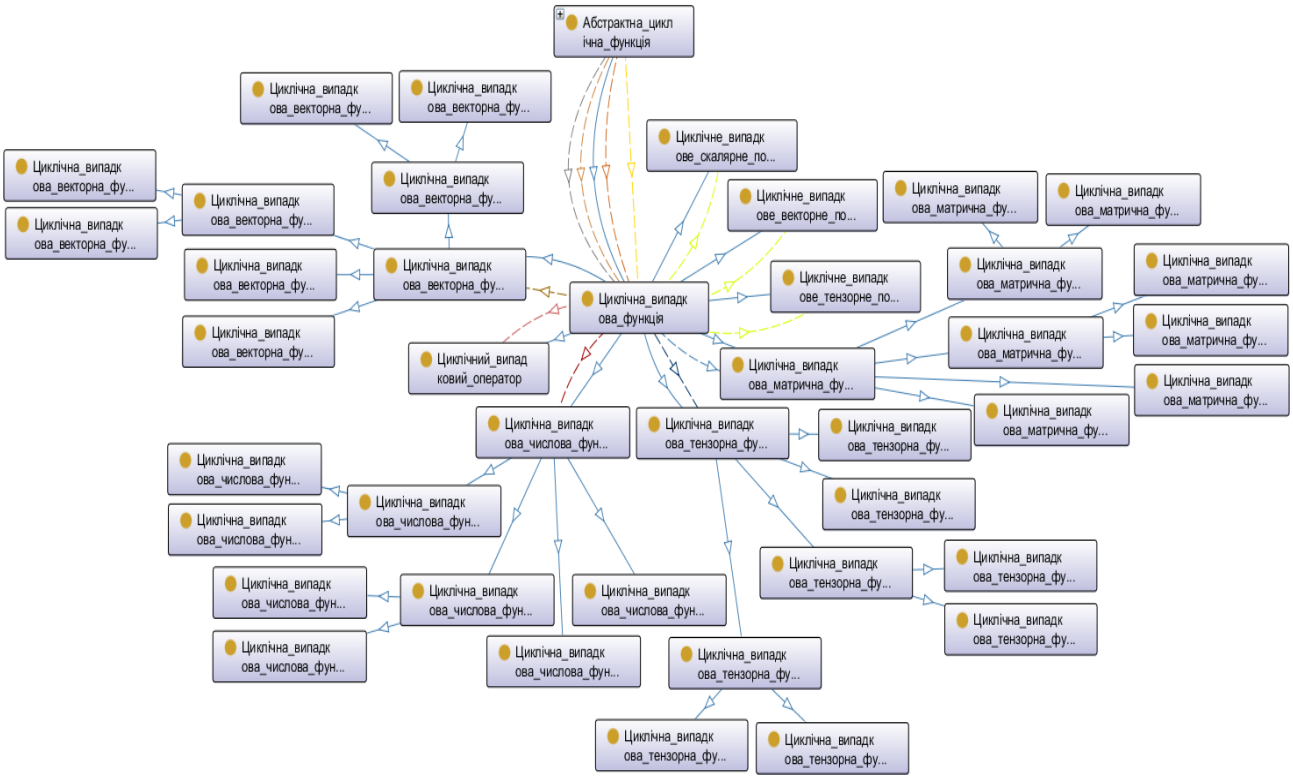

Рис. 11. Циклічна випадкова функція

3. Встановлення та налаштування програмних засобів

Для забезпечення роботи електронного підручника потрібно встановити та налаштувати наступне ПЗ: Node.js, npm, MongoDB, Heroku та Git.

Node.js – платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Щоб встановити програму необхідно на веб-сайті (https://nodejs.org/uk/) завантажити інсталятор, як це показано на рис. 12.

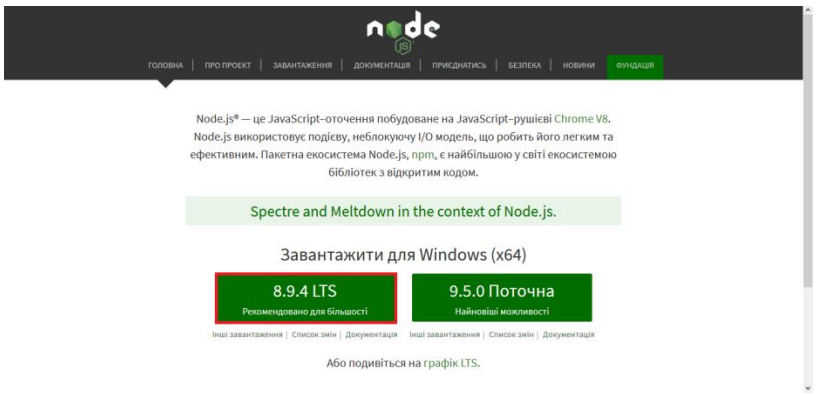

Рис. 12. Веб-сайт Node.js

Далі потрібно запустити на виконання файл інсталяції (на час розробки остання стабільна версія – node-v8.9.4-x64.exe). У вітальному вікні потрібно натиснути кнопку “Next”, далі, після ознайомлення з ліцензійною угодою, необхідно встановити прапорець про прийняття положень цієї угоди і натиснути “Next”. Наступним вікно є вибір папки для встановлення (рис. 13), її слід залишити за замовчуванням і натиснути “Next”. Далі можна вибрати, які компоненти слід встановити (рекомендовано залишити без змін). В останньому вікні потрібно натиснути кнопку “Install”, після чого Node.js буде встановлено на комп’ютер.

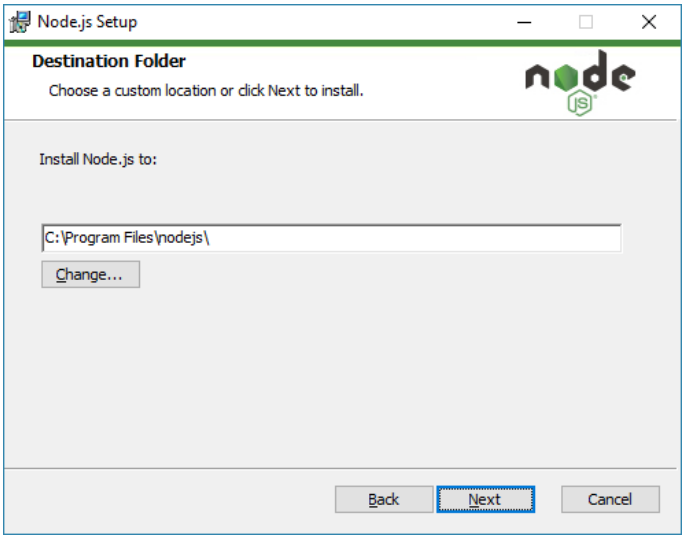

Рис. 13. Вікно вибору папки встановлення Node.js

Npm це менеджер пакунків для мови програмування JavaScript. В Node.js є менеджером пакунків за замовчуванням, тому не потрібно завантажувати його окремо.

MongoDB — документо-орієнтована система керування базами даних з відкритим сирцевим кодом, яка не потребує опису схеми таблиць. На

офіційному веб-сайті (https://www.mongodb.com/) потрібно завантажити інсталятор, як показано на рис. 14. Тоді в центрі завантажень MongoDB потрібно обрати “Community Server” та ОС відповідно до системи (під час розробки використано ОС Windows), після цього натиснути кнопку “DOWNLOAD (msi)” (рис. 15).

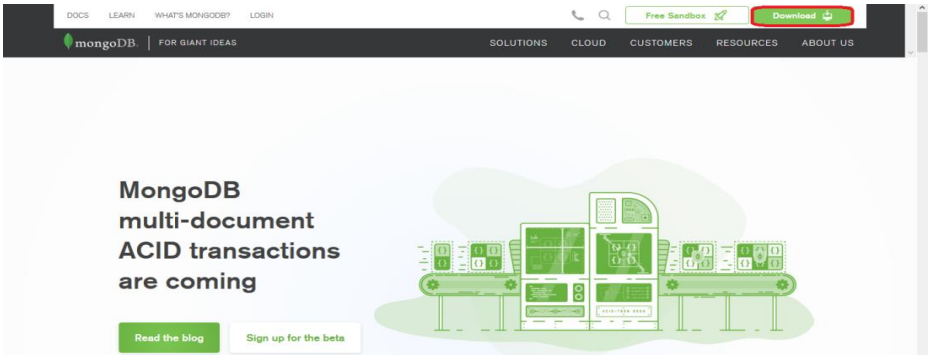

Рис. 14. Веб-сайт MongoDB

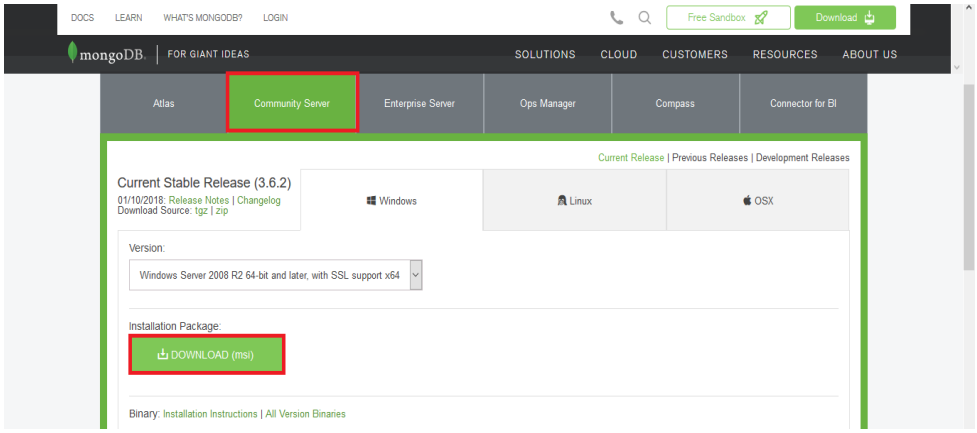

Рис. 15. Сторінка завантаження MongoDB

Далі потрібно запустити інсталятор (mongodb-win32-x86_64-2008plus-ssl-3.6.2-signed.exe). Після вітального вікна потрібно ознайомитися та прийняти ліцензійну угоду. У наступному вікні потрібно вибрати тип інсталяції “Complete”, як це показано на рис. 16. Тоді, інсталятор запропонує встановити графічну оболонку MongoDB Compass, що, в ході розробки БД до програмної системи не використовується, тому потрібно зняти галочку біля “Install MongoDB Compass” (рис. 17).

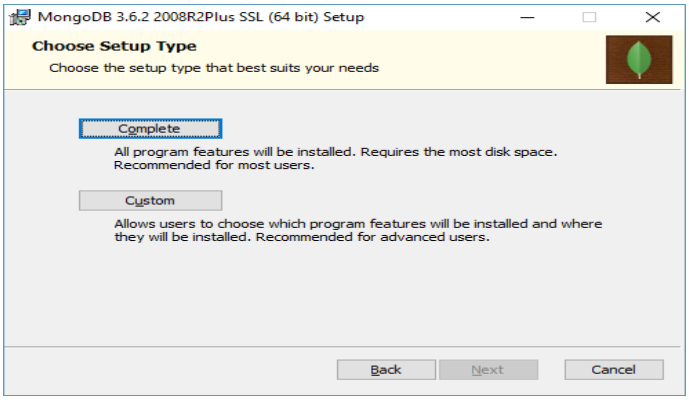

Рис. 16. Варіанти інсталяції MongoDB

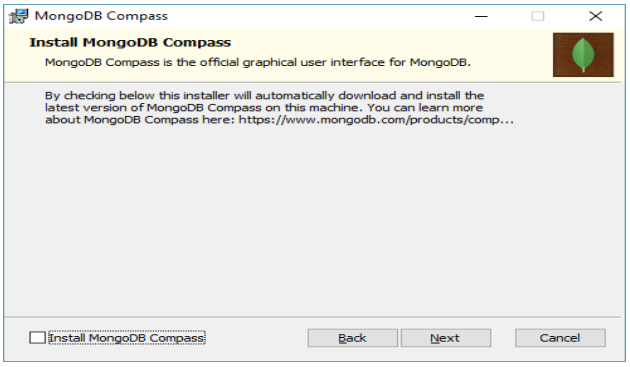

Рис. 17. Вікно встановлення MongoDB Compass

В останньому вікні потрібно натиснути “Install”. Після чого виконається процес встановлення MongoDB.

Рекомендовано встановити MongoDB в якості сервісу Windows, для зручного керування програмою. Для цього потрібно в папці MongoDB\Server\3.6 створити папку data в якій створити папку для БД (db) та журналу (log), як це показано на рис. 18. Щоб створити папку в ОС Windows потрібно викликати контекстне меню правою кнопкою миші, та вибрати “Створити > Папку”.

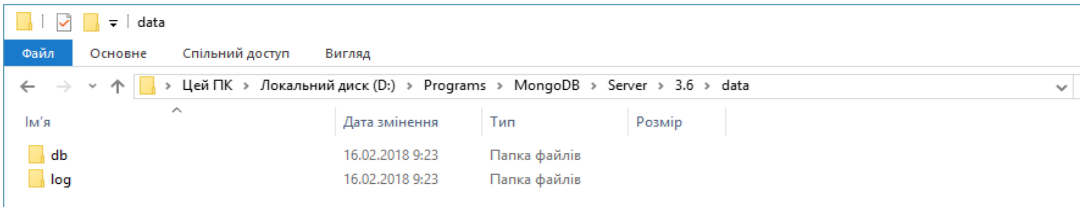

Рис. 18. Створення необхідних папок для MongoDB

Далі необхідно створити конфігураційний файл mongod.cfg в папці MongoDB\Server\3.6, як показано на рис. 19.

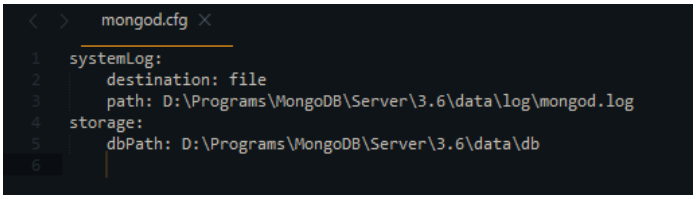

Рис. 19. Конфігураційний файл mongod.cfg

Далі потрібно відкрити командний рядок від імені адміністратора – натиснути комбінацію клавіш “Windows + R”, у відкритому вікні (рис. 20) ввести “cmd” та натиснути комбінацію “Ctrl + Shift + Enter”.

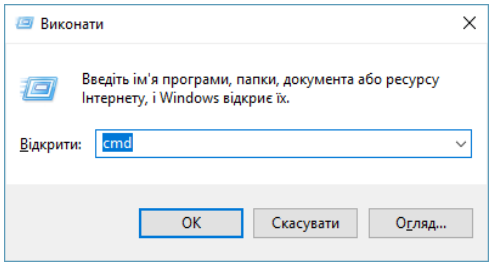

Рис. 20. Запуск командного рядка

Інсталяція MongoDB сервісу виконується при запуску виконуваного файлу (mongod.exe) з опціями --install та --config (mongod.cfg). Отже, в командний рядок потрібно вести наступну команду:

```
"D:\Programs\MongoDB\Server\3.6\bin\mongod.exe" --config
"D:\Programs\MongoDB\Server\3.6\mongod.cfg" –install
```

Далі, потрібно запустити MongoDB сервіс, як показано на рис. 21.

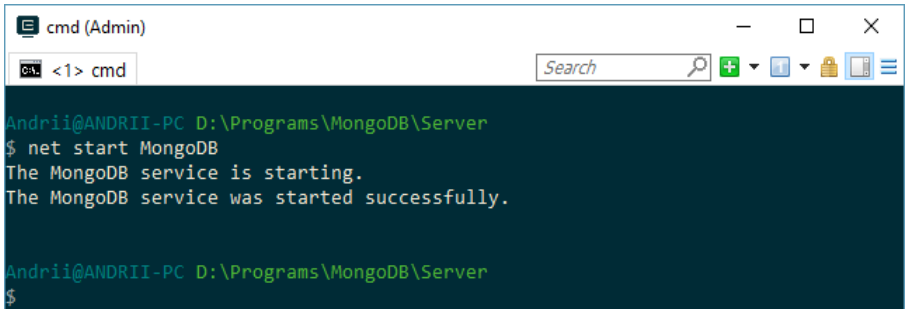

Рис. 21. Запуск сервісу MongoDB

Щоб перевірити роботу MongoDB потрібно виконати дії, показані на рис. 22.

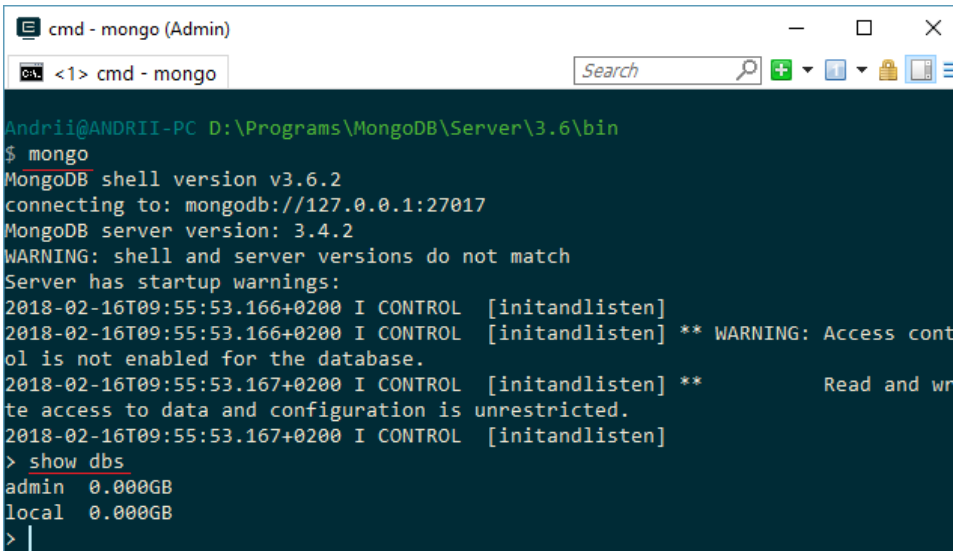

Рис. 22. Перевірка роботи MongoDB

Для розміщення програмної систему в Інтернеті використано хмарний сервіс Heroku. Щоб встановити дану програму потрібно перейти на веб-сайт “Heroku Dev Center” – https://devcenter.heroku.com/start (рис.23) та вибрати технологію Node.js.

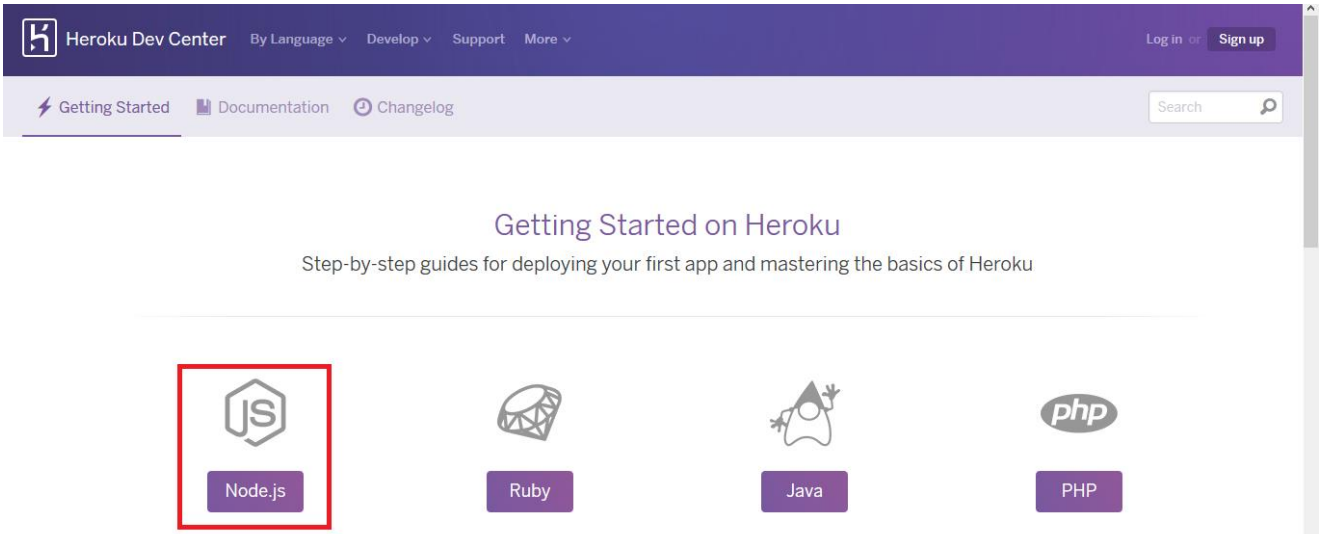

Рис. 23. "Heroku Dev Center"

В розділі "Set up" натиснути кнопку "Download the Heroku CLI for…" та вибрати ОС (розробку виконано в ОС Windows). Далі виконати файл інсталяції (heroku-windows-amd64.exe). В першому вікні можна вибрати компоненти (рис. 24), в другому папку для встановлення (рекомендовано залишити параметри за замовчуванням).

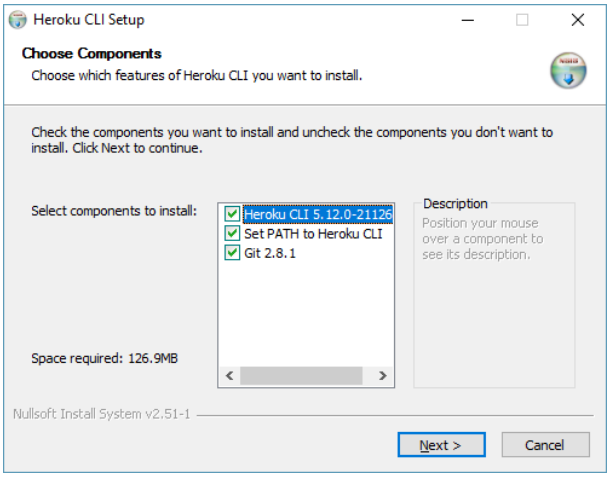

Рис. 24. Вікно вибору компонентів Heroku

Оскільки, Heroku використовує технологію Git, то її включено за замовчуванням. Під час встановлення, відкриється вікно інсталяції Git, як це показано на рис. 25. Рекомендовано не змінювати параметри, доступні, при встановленні Git.

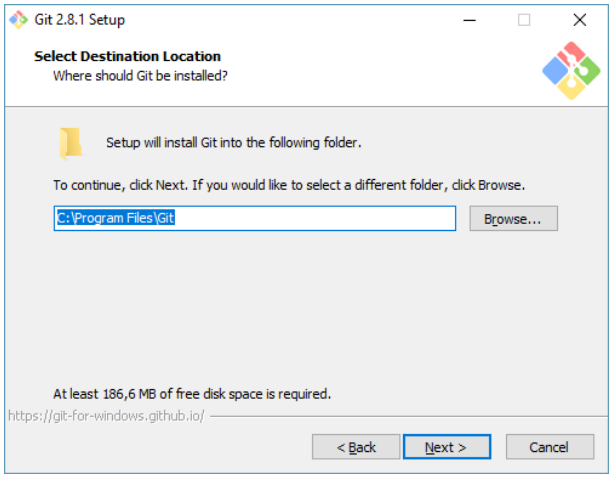

Рис. 25. Вікно інсталяції Git

Після встановлення Heroku та Git, в командному рядку (CMD) потрібно авторизуватися в системі, як показано на рис. 26.

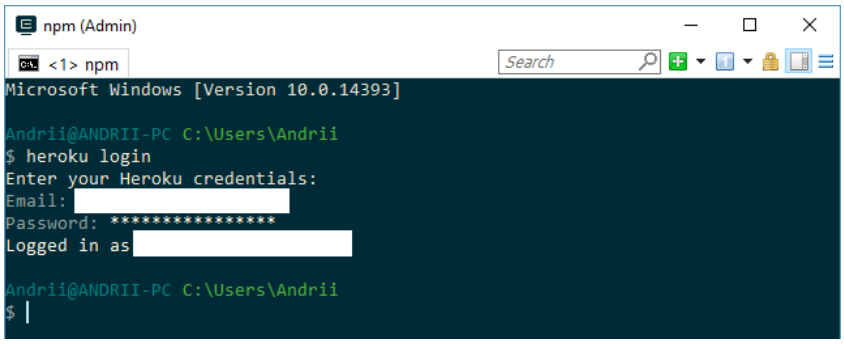

Рис. 26. Вхід в систему Heroku

Для роботи з Heroku потрібно переконатися що всі необхідні програми встановлено, тому потрібно виконати команди, показані на рис. 27.

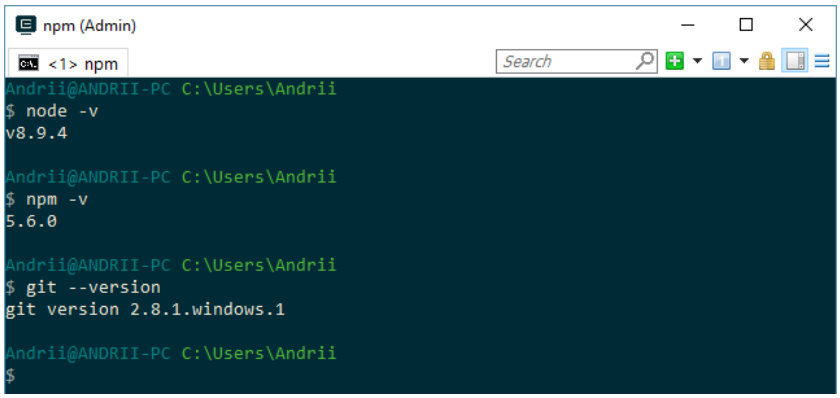

Рис. 27. Перевірка наявності необхідних програм для роботи з Heroku

Для завантаження програмної системи на хмарний сервіс Heroku необхідно:

– створити проект на сервісі (heroku create);

– завантажити код (git push heroku master);

– запустити мінімум один екземпляр програми (heroku ps:scale web=1);

– запустити веб-сайт в браузері (heroku open).

Програма Protege Desktop 5.0 поширюється у вигляді ZIP-файлу на головному веб-сайті Protégé (рис. 28) і включає 64-розрядне середовище

виконання Java (JRE). Тому, не потрібно встановлювати Java на комп'ютері, для запуску середовища.

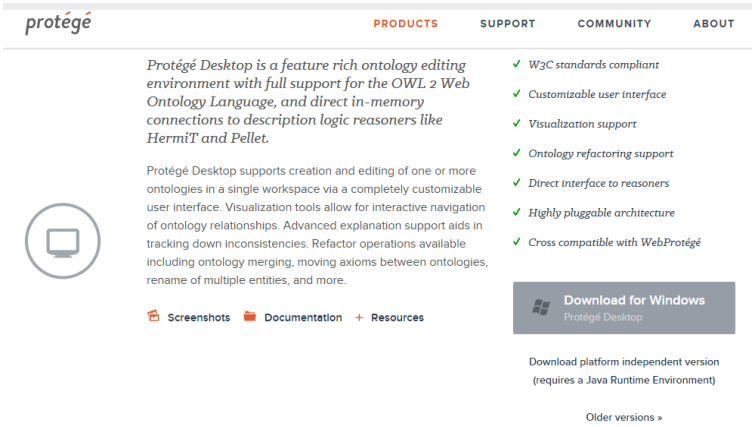

Рис. 28. Веб-сайт Protégé

Щоб завантажити ZIP-файл на комп'ютер потрібно виконати клік по кнопці "Download for Windows" на головній сторінці веб-сайту Protégé. Після завершення завантаження потрібно перейти до папки, в яку був завантажений ZIP-файл. На комп'ютері з ОС Windows це, як правило, буде папка "Downloads". Далі потрібно виконати клік правою кнопкою миші по ZIP-файлу та обрати "Видобути все..." з контекстного меню, щоб відкрити діалогове вікно приведене на рис. 29.

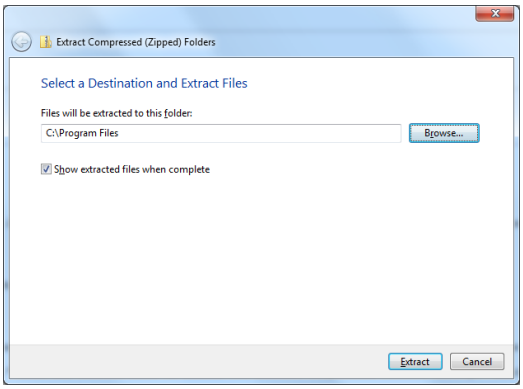

Рис. 29. Діалогове вікно для видобування файлів з архіву

Далі потрібно натиснути кнопку "Browse...", щоб вибрати місце для встановлення Protege Desktop. Protege Desktop буде видобуто до обраного пункту призначення в папку під назвою "Protege_5.0.0" (рис. 30).

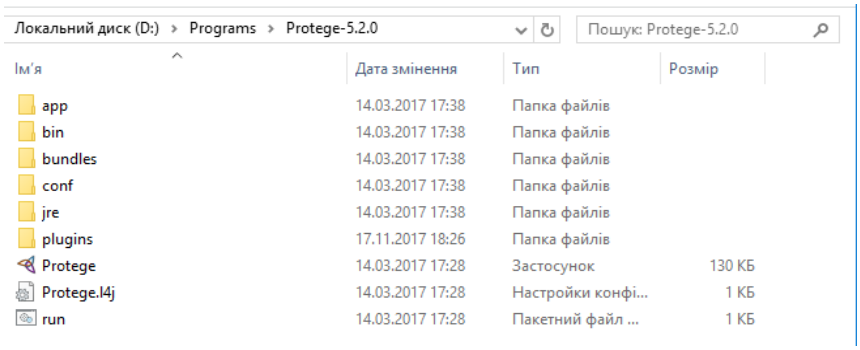

Рис. 30. Папка із файлами програми Protégé

Існує два способи запуску Protégé:

– подвійне клацання по фалу Protégé.exe;

– подвійне клацання по файлу run.bat, який запустить Protégé і консоль.

Для більш швидкого доступу до Protege Desktop можна, за допомогою правої кнопки миші, викликати контекстне меню файлу Protege.exe і вибрати “Надіслати” на “Робочий стіл (створити ярлик)”. Це створить ярлик для запуску Protege Desktop на робочому столі Windows.

4. Встановлення та налаштування електронного підручника

Електронний підручник слід встановлювати після інсталяції необхідного програмного забезпечення: Node.js, Npm, MongoDB, Heroku, Git.

У директорії програмної системи (laboratory) потрібно викликати командний рядок та ввести команду для завантаження та встановлення необхідних бібліотеки, як це показано на рис. 31. Після цього в директорії буде автоматично створено папку node_modules, в якій будуть доступні усі бібліотеки.

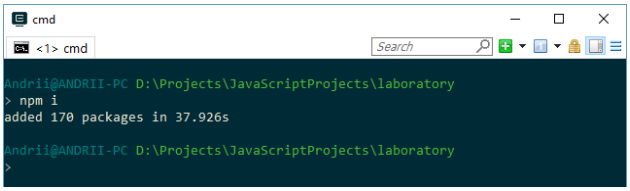

Рис. 31. Встановлення бібліотек програмної системи

Встановлення відбувається за допомогою менеджера пакунків Npm, програма зчитує дані з файлу package.json. Також, необхідно окремо додати бібліотеку Nodemon командою npm -i nodemon -g. Запуск програмної системи здійснюється командою npm run server, як показано на рис. 32.

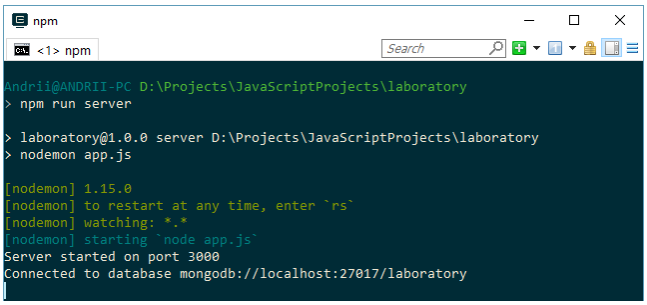

Рис. 32. Запуск програмної системи

Далі необхідно запустити браузер, і в адресному рядку ввести http://localhost:3000/ (рис. 33).

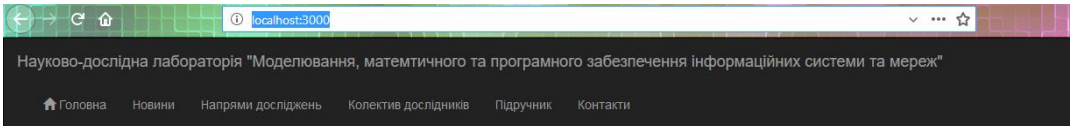

Рис. 33. Програмна система

Таким чином, програмну систему встановлено та налаштовано для роботи в локальному середовищі.

ЛАБОРАТОРНА РОБОТА №4

Тема. Генетичні алгоритми.

Мета. Навчитися використовувати генетичні алгоритми для вирішення задач.

1. КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Генетичні алгоритми виникли в результаті спостереження і спроб копіювання природних процесів, що відбуваються в світі живих організмів, зокрема, еволюції та пов’язаної з нею селекції (природного відбору) популяцій живих істот.

Генетичний алгоритм - це методологія вирішення проблем на основі штучного інтелекту. Це нематематичний, недетермінований, але стохастичний процес або алгоритм для вирішення проблем оптимізації.

Генетичні алгоритми застосовуються при розробці програмного забезпечення, в системах штучного інтелекту, оптимізації, штучних нейронних мережах і в інших галузях знань. Слід зазначити, що з їх допомогою вирішуються завдання, для яких раніше використовувалися тільки нейронні мережі. У цьому випадку генетичні алгоритми виступають просто в ролі незалежного від нейронних мереж альтернативного методу, призначеного для вирішення тієї ж самої задачі.

Генетичний алгоритм являє собою метод, що відображає природну еволюцію методів вирішення проблем, і в першу чергу задач оптимізації. Генетичні алгоритми — це процедури пошуку, засновані на механізмах природного відбору і спадкоємства. У них використовується еволюційний принцип виживання найбільш пристосованих особин. Вони відрізняються від традиційних методів оптимізації декількома базовими елементами. Зокрема, генетичні алгоритми:

- обробляють не значення параметрів самого завдання, а їх закодовану форму;

- здійснюють пошук рішення виходячи не з єдиної точки, а з їх деякої популяції;

- використовують тільки цільову функцію, а не її похідні або іншу додаткову інформацію;

- застосовують імовірнісні, а не детерміновані правила вибору.

Основний (класичний) генетичний алгоритм (який також називається елементарним чи простим генетичним алгоритмом) складається з наступних кроків:

- ініціалізація, або вибір вихідної популяції хромосом;
- оцінка пристосованості хромосом в популяції;
- перевірка умови зупинки алгоритму;
- селекція хромосом;
- застосування генетичних операторів;
- формування нової популяції;
- вибір “найкращої” хромосоми.

2. ПРАКТИЧНА ЧАСТИНА

Розглянемо діофантове (тільки цілі рішення) рівняння: 1a+2b+3c+4d=30, де a, b, c і d - деякі додатні цілі числа. Застосування генетичного алгоритму за дуже короткий час знаходить шуканий розв'язок (a, b, c, d).

1. Спочатку потрібно вибрати 5 випадкових рішень (a,b,c,d), менше 30.

Хромосома	(a,b,c,d)
1	(7, 6, 2, 2)
2	(6, 3, 6, 6)
3	(4, 2, 8, 3)
4	(1, 8, 5, 1)
5	(6, 2, 8, 2)

2. Для того, щоб визначити коефіцієнти пристосування необхідно підставити кожне рішення у рівняння 1a+2b+3c+4d=30. Відстань від отриманого значення до 30 і буде потрібним значенням.

Хромосома	Коефіцієнти пристосованості
1	|33 - 30| = 3
2	|54 - 30| = 24
3	|44 - 30| = 14

4	|36 - 30| = 6
5	|42 - 30| = 12

3. Оскільки менші значення ближчі до 30, то вони більш бажані. Щоб створити систему, де хромосоми з більш придатними значеннями мають великі шанси виявитися батьками, необхідно обчислити з якою ймовірністю (в %) може бути обрана кожна з них. Далі необхідно взяти суму зворотних значень коефіцієнтів, і виходячи з цього обчислювати відсотки.

Сума зворотних значень коефіцієнтів обчислюється наступним чином:

$$(1/3)+(1/24)+(1/14)+(1/6)+(1/12) = 0.6964$$

Хромосома	Пристосованість
1	(1/3)/0.6964=47,86%
2	(1/24)/0.6964=5,98%
3	(1/14)/0.6964=10,26%
4	(1/6)/0.6964=23,93%
5	(1/12)/0.6964=11,97%

4. Відповідно до пристосованості хромосом можна обрати хромосоми - батьки.

Хромосома-батько	Хромосома-мати
4	3
3	4
1	4
3	1
4	1

5. Кожен нащадок містить інформацію про гени батька і матері. Є декілька способів схрещування хромосом (кросинговерів).

Хромосома-батько	Хромосома-мати	Хромосома-нащадок
$a_1 \mid b_1, c_1, d_1$	$a_2 \mid b_2, c_2, d_2$	a_1,b_2, c_2, d_2 або a_2,b_1, c_1, d_1
$a_1,b_1 \mid c_1, d_1$	$a_2,b_2 \mid c_2, d_2$	a_1,b_1, c_2, d_2 або a_2,b_2, c_1, d_1
$a_1, b_1, c_1 \mid d_1$	$a_2, b_2, c_2 \mid d_2$	a_1,b_1, c_1, d_2 або a_2,b_2, c_2, d_1

Хромосома-батько	Хромосома-мати	Хромосома-нащадок
(1, 8, 5, 1)	(4, 2, 8, 3)	(1, 2, 8, 3)
(4, 2, 8, 3)	(1, 8, 5, 1)	(1, 8, 8, 8)
(7, 6, 2, 2)	(1, 8, 5, 1)	(1, 8, 2, 2)
(4, 2, 8, 3)	(7, 6, 2, 2)	(4, 6, 2, 2
(1, 8, 5, 1)	(7, 6, 2, 2)	(7, 6, 5, 1)

6. Наступним кроком є обчислення коефіцієнтів пристосування хромосом - нащадків. Підставивши в рівняння 1a+2b+3c+4d=30 значення хромосом-нащадків визначаємо коефіцієнти пристосованості. Вже у другому поколінні одна з хромосом досягла коефіцієнта пристосованості 0, тобто стала рішенням задачі.

Хромосома-нащадок	Коефіцієнти пристосованості
(1, 2, 8, 3)	|41 - 30| = 11
(1, 8, 8, 3)	|53 - 30| = 23
(1, 8, 2, 2)	|31 - 30| = 1
(4, 6, 2, 2)	|30 - 30| = 0
(7, 6, 5, 1)	|38 - 30| = 8

Нижче наведено фрагмент коду програми мовою Java, яка розв'язує рівняння згідно алгоритму (повний лістинг програми можна знайти у файлообміннику):

```
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        if (actionEvent.getSource() == Res) {
            step = 0;
            return;
        }
        if (step == 0 && !testFields()) return;
        if (step == 0) {
            for (int i = 0; i < randNum.length; i++) {
                for (int j = 0; j < randNum[i].length; j++) {
                    randNum[i][j] = (int) Math.round(Math.random() * ((fields[4] / 3) - 2) + 1);
                }
            }
        }
        System.out.println("Покоління " + (++step));
        for (int i = 0; i < randNum.length; i++) {
            System.out.println(i + 1 + "  |  " + Arrays.toString(randNum[i]));
        }
        System.out.println();
        randExFi = new int[]{0, 0, 0, 0, 0};
        for (int i = 0; i < randNum.length; i++) {
            for (int j = 0; j < randNum[i].length; j++) {
                randExFi[i] += (fields[j] * randNum[i][j]);
            }
            randExRe[i] = Math.abs(randExFi[i] - fields[4]);
            System.out.println(i + 1 + " | |" + randExFi[i] + " - " + fields[4] + "| = " + randExRe[i]);
        }
```

```
        System.out.println();
        double percent = (1.0 / randExRe[0]) + (1.0 / randExRe[1]) + (1.0 / randExRe[2]) +

(1.0 / randExRe[3] + (1.0 / randExRe[4]));
        for (int i = 0; i < randExPc.length; i++) {
            randExPc[i] = ((1.0 / randExRe[i]) / percent) * 100;
            System.out.println(i + 1 + " | (1 / " + randExRe[i] + ") / " + percent + " * 100 = " + randExPc[i]);
        }
        System.out.println();
        int max = 0, min = 0;
        for (int i = 0; i < randExPc.length; i++) {
            if (randExPc[min] > randExPc[i]) {
                min = i;
            }
            if (randExPc[max] < randExPc[i]) {
                max = i;
            }
        }
        for (int i = 0; i < randFx.length; i++) {
            randFx[i] = (int) Math.round(Math.random() * 4);
            if (randFx[i] == min) randFx[i] = max;
        }
        for (int i = 0; i < randMx.length; i++) {
            do {
                randMx[i] = (int) Math.round(Math.random() * 4);
                if (randMx[i] == min) randMx[i] = max;
            } while (randFx[i] == randMx[i]);
```

```
            System.out.println(randMx[i] + 1 + " | " + (randFx[i]
+ 1));
        }
        System.out.println();
        for (int i = 0; i < randRes.length; i++) {
            switch ((int) Math.round(Math.random() * 5)) {
                case 0:
                    randRes[i][0] = randNum[randFx[i]][0];
                    randRes[i][1] = randNum[randMx[i]][1];
                    randRes[i][2] = randNum[randMx[i]][2];
                    randRes[i][3] = randNum[randMx[i]][3];
                    break;
                case 1:
                    randRes[i][0] = randNum[randMx[i]][0];
                    randRes[i][1] = randNum[randFx[i]][1];
                    randRes[i][2] = randNum[randFx[i]][2];
                    randRes[i][3] = randNum[randFx[i]][3];
                    break;
                case 2:
                    randRes[i][0] = randNum[randFx[i]][0];
                    randRes[i][1] = randNum[randFx[i]][1];
                    randRes[i][2] = randNum[randMx[i]][2];
                    randRes[i][3] = randNum[randMx[i]][3];
                    break;
                case 3:
                    randRes[i][0] = randNum[randMx[i]][0];
                    randRes[i][1] = randNum[randMx[i]][1];
                    randRes[i][2] = randNum[randFx[i]][2];
                    randRes[i][3] = randNum[randFx[i]][3];
```

```
                    break;
                case 4:
                    randRes[i][0] = randNum[randMx[i]][0];
                    randRes[i][1] = randNum[randMx[i]][1];
                    randRes[i][2] = randNum[randMx[i]][2];
                    randRes[i][3] = randNum[randFx[i]][3];
                    break;
                case 5:
                    randRes[i][0] = randNum[randFx[i]][0];
                    randRes[i][1] = randNum[randFx[i]][1];
                    randRes[i][2] = randNum[randFx[i]][2];
                    randRes[i][3] = randNum[randMx[i]][3];
                    break;
            }

System.out.println(Arrays.toString(randNum[randMx[i]]) + " | " +
                    Arrays.toString(randNum[randFx[i]]) + " | " +
                    Arrays.toString(randRes[i]));
        }
        System.out.println();
        for (int i = 0; i < randNum.length; i++) {
            System.arraycopy(randRes[i],   0,   randNum[i],   0,
randNum[i].length);
        }
    }
```

3. ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Ознайомитись з теоретичними відомостями до лабораторної роботи.

2. Завантажити вихідний код з файлообмінника.

3. Виконати індивідуальне завданням згідно варіанту.

4. Оформити звіт.

5. Зробити висновок згідно отриманих результатів.

Завдання

Варіант	Завдання
1	a+4b+2c+5d=50
2	2a+7b+c+4d=40
3	2a+5b+3c+d=60
4	3a+4b+c+4d=40
5	5a+b+6c+2d=50
6	a+2b+6c+9d=40
7	3a+4b+2c+7d=60
8	4a+5b+7c+d=70
9	6a+3b+2c+3d=50
10	7a+2b+5c+3d=70

ЛАБОРАТОРНА РОБОТА №5

Тема: Проектування та навчання штучної нейронної мережі для задач класифікації цифр.

Мета: Вивчити принципи проектування, навчання штучної нейронної мережі для задач класифікації цифр.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Ознайомившись з матеріалом в двох попередніх лекціях, тепер напишемо програму, що навчається розпізнаванню рукописних цифр з використанням стохастичного градієнтного спуску і навчальних даних від MNIST. Зробимо це за допомогою короткої програми на python 2.7, що складається всього з 74 рядків. Перше, що нам потрібно - завантажити дані MNIST з файлообмінника курсу.

Крім даних MNIST нам для швидких обчислень лінійної алгебри також знадобиться бібліотека для **python** під назвою **Numpy**. Якщо її у вас немає, ви можете взяти її за посиланням.

Перед тим, як перейти до коду всієї програми, розглянемо основні особливості коду для нейронної мережі. Центральне місце займає клас **Network**, який використовується для подання нейронної мережі. Ось код ініціалізації об'єкта **Network**:

```
class Network(object):


    def __init__(self, sizes):
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in
sizes[1:]]
        self.weights = [np.random.randn(y, x)
                        for x, y in zip(sizes[:-1],
sizes[1:])]
```

Масив sizes містить кількість нейронів у відповідних шарах. Так що, якщо ми хочемо створити об'єкт **Network** з двома нейронами в першому шарі, трьома нейронами в другому шарі, і одним нейроном в третьому, то ми запишемо так:

```
net = Network([2, 3, 1])
```

Зміщення і ваги в об'єкті **Network** ініціалізуються випадковим чином з використанням функції **np.random.randn** з **Numpy**, яка генерує **розподіл Гаусса** з математичним очікуванням 0 і середньоквадратичним відхиленням 1. Така випадкова ініціалізація дає алгоритму стохастичного градієнтного спуску відправну точку. Код ініціалізації **Network** передбачає, що перший шар нейронів буде вхідним, і не призначає їм зміщення, оскільки вони використовуються лише при підрахунку вихідних даних.

Слід відзначити, що зміщення і ваги зберігаються як масив матриць **Numpy**. Наприклад, **net.weights[1]** - матриця **Numpy**, що зберігає ваги, що з'єднують другий і третій шари нейронів (це не перший і другий шари, оскільки в **python** нумерація елементів масиву йде з нуля). Позначимо матрицю **net.weights[1]** , як *w*. Це така матриця, що w_{jk} - це вага зв'язку між *k*-м нейроном у другому шарі і *j*-м нейроном в третьому. Такий порядок індексів *j* і *k* може здатися дивним - не було б логічніше поміняти їх місцями? Але великою перевагою такого запису є те, що вектор активацій третього шару нейронів рівний:

$$a' = \sigma(wa + b) \tag{1}$$

Давайте розберемо це рівняння. *a* - вектор активацій другого шару нейронів. Щоб отримати *a'*, ми множимо *a* на матрицю ваг *w*, і додаємо вектор зміщень *b*. Потім ми застосовуємо сигмоїду σ поелементно до кожного елементу вектора *wa* + *b* (це називається векторизацією функції σ). Легко перевірити, що рівняння (1) дає такий же результат, що і правило (2) для обчислення сигмоподібного нейрона.

$$\frac{1}{1 + exp(-\sum_i w_i \cdot x_i - b)}. \tag{2}$$

З огляду на все це, легко написати код, який обчислює вихідні дані об'єкта **Network**. Почнемо з визначення сигмоїди:

```
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))
```

Врахуйте, що коли параметр *z* буде вектором або масивом **Numpy**, то **Numpy** автоматично буде застосовувати сигмоїду поелементно, тобто, в векторному вигляді.

Додамо метод прямого поширення в клас **Network**, який приймає на вхід *a* від мережі і повертає відповідні вихідні дані. Передбачається, що параметр *a* - це (n, 1) **Numpy ndarray**, а не вектор *(n,)*. Тут *n* - кількість вхідних нейронів. Якщо ви спробуєте використовувати вектор *(n,)*, то отримаєте дивні результати.

Метод просто застосовує рівняння (1) до кожного шару:

```
    def feedforward(self, a):
        """Повернути вихідні дані мережі при вхідних даних "a""""
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a)+b)
        return a
```

Звичайно, в основному нам від об'єктів **Network** потрібно, щоб вони навчалися. Для цього ми дамо їм метод SGD, який реалізує стохастичний градієнтний спуск. Ось його код. У деяких місцях він досить загадковий, але нижче ми розберемо його докладніше.

```
    def SGD(self, training_data, epochs, mini_batch_size, eta,
            test_data=None):
        """Навчаємо мережу за допомогою міні-пакетів і стохастичного градієнтного спуску. training_data - список кортежів "(x, y)", що позначають навчальні вхідні дані і бажані вихідні. Решта обов'язкові параметри говорять самі за себе. Якщо test_data заданий, тоді мережа буде оцінюватися щодо даних перевірки після кожної епохи, і буде виводитися поточний прогрес. Це корисно для відстеження прогресу, проте істотно сповільнює роботу."""
        if test_data: n_test = len(test_data)
        n = len(training_data)
        for j in xrange(epochs):
            random.shuffle(training_data)
            mini_batches = [
                training_data[k:k+mini_batch_size]
                for k in xrange(0, n, mini_batch_size)]
            for mini_batch in mini_batches:
                self.update_mini_batch(mini_batch, eta)
```

```
            if test_data:
                print "Epoch {0}: {1} / {2}".format(
                    j, self.evaluate(test_data), n_test)
            else:
                print "Epoch {0} complete".format(j)
```

training_data - список кортежів "(x, y)", що позначають навчальні вхідні дані і бажані вихідні. Змінні **epochs** і **mini_batch_size** - це кількість епох для навчання і розмір міні-пакетів для використання. **eta** - швидкість навчання, η. Якщо **test_data** заданий, тоді мережа буде оцінюватися щодо перевірочних даних після кожної епохи, і буде виводитися поточний прогрес. Це корисно для відстеження прогресу, проте істотно сповільнює роботу.

Код працює так. В кожну епоху він починає з того, що випадково перемішує навчальні дані, а потім розбиває їх на міні-пакети потрібного розміру. Це простий спосіб створення вибірки з навчальних даних. Потім для кожного **mini_batch** застосовуємо один крок градієнтного спуску. Це робить код **self.update_mini_batch(mini_batch, eta)**, який оновлює ваги і зміщення мережі відповідно до однієї ітерації градієнтного спуску, використовуючи тільки навчальні дані в **mini_batch**. Код для методу **update_mini_batch**:

```
    def update_mini_batch(self, mini_batch, eta):
        """Оновити ваги і зміщення мережі, застосовуючи
градієнтний спуск з використанням зворотного поширення до одного
міні-пакету. mini_batch - це список кортежів (x, y), а eta -
швидкість навчання."""
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        for x, y in mini_batch:
            delta_nabla_b, delta_nabla_w = self.backprop(x,
y)
            nabla_b = [nb+dnb for nb, dnb in zip(nabla_b,
delta_nabla_b)]
            nabla_w = [nw+dnw for nw, dnw in zip(nabla_w,
delta_nabla_w)]
```

```
            self.weights = [w-(eta/len(mini_batch))*nw
                            for w, nw in zip(self.weights,
nabla_w)]
            self.biases = [b-(eta/len(mini_batch))*nb
                           for b, nb in zip(self.biases,
nabla_b)]
```

Більшу частину роботи робить рядок:

```
delta_nabla_b, delta_nabla_w = self.backprop(x, y)
```

Він викликає алгоритм зворотного поширення - це швидкий спосіб обчислити градієнт функції вартості. Так що **update_mini_batch** просто обчислює ці градієнти для кожного навчального прикладу з **mini_batch**, а потім оновлює **self.weights** і **self.biases**.

Розглянемо програму цілком, включаючи пояснювальні коментарі. За винятком функції **self.backprop** програма говорить сама за себе - основну роботу роблять **self.SGD** і **self.update_mini_batch**. Метод **self.backprop** використовує кілька додаткових функцій для обчислення градієнта, а саме, **sigmoid_prime**, яка обчислює похідну сигмоїди, і **self.cost_derivative**, яку ми розглядати не будемо. Ви можете отримати про них уявлення, вивчивши код і коментарі. Врахуйте, що, хоча програма здається довгою, велика частина коду - це коментарі, які полегшують розуміння. Весь код доступний у файлообміннику.

```
"""
network.py
~~~~~~~~~~


Модуль реалізації навчального алгоритму стохастичного
градієнтного спуску для нейронної мережі прямого поширення.
Градієнти обчислюються за допомогою зворотного поширення.
"""


#### Бібліотеки
# Стандартна бібліотека
import random
```

```
    # Сторонні бібліотеки
    import numpy as np


    class Network(object):


        def __init__(self, sizes):
            """Масив sizes містить кількість нейронів у відповідних шарах. Так що, якщо ми хочемо створити об'єкт Network з двома нейронами в першому шарі, трьома нейронами в другому шарі, і одним нейроном в третьому, то ми запишемо це, як [2, 3, 1]. Зміщення і ваги мережі започатковано випадковим чином з використанням розподілу Гаусса з математичним очікуванням 0 і середньоквадратичним відхиленням 1. Передбачається, що перший шар нейронів буде вхідним, і тому у його нейронів немає зсувів, оскільки вони використовуються лише при підрахунку вихідних даних.
"""
            self.num_layers = len(sizes)
            self.sizes = sizes
            self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
            self.weights = [np.random.randn(y, x)
                            for x, y in zip(sizes[:-1], sizes[1:])]


        def feedforward(self, a):
            """Повертає вихідні дані мережі, коли a - вхідні дані."""
            for b, w in zip(self.biases, self.weights):
                a = sigmoid(np.dot(w, a)+b)
            return a


        def SGD(self, training_data, epochs, mini_batch_size, eta,
```

```
            test_data=None):

        """Навчаємо мережу за допомогою міні-пакетів і
стохастичного градієнтного спуску. training_data - список кортежів
"(x, y)", що позначають навчальні вхідні дані і бажані вихідні.
Решта обов'язкові параметри говорять самі за себе. Якщо test_data
заданий, тоді мережа буде оцінюватися щодо даних перевірки після
кожної епохи, і буде виводитися поточний прогрес. Це корисно для
відстеження прогресу, проте істотно сповільнює роботу."""

        if test_data: n_test = len(test_data)

        n = len(training_data)

        for j in xrange(epochs):

            random.shuffle(training_data)

            mini_batches = [

                training_data[k:k+mini_batch_size]

                for k in xrange(0, n, mini_batch_size)]

            for mini_batch in mini_batches:

                self.update_mini_batch(mini_batch, eta)

            if test_data:

                print "Epoch {0}: {1} / {2}".format(

                    j, self.evaluate(test_data), n_test)

            else:

                print "Epoch {0} complete".format(j)


    def update_mini_batch(self, mini_batch, eta):

        """Оновити ваги і зміщення мережі, застосовуючи
градієнтний спуск з використанням зворотного поширення до одного
міні-пакету. mini_batch - це список кортежів (x, y), а eta -
швидкість навчання. """

        nabla_b = [np.zeros(b.shape) for b in self.biases]

        nabla_w = [np.zeros(w.shape) for w in self.weights]

        for x, y in mini_batch:

            delta_nabla_b, delta_nabla_w = self.backprop(x,
y)
```

```
                nabla_b = [nb+dnb for nb, dnb in zip(nabla_b,
delta_nabla_b)]
                nabla_w = [nw+dnw for nw, dnw in zip(nabla_w,
delta_nabla_w)]
            self.weights = [w-(eta/len(mini_batch))*nw
                            for w, nw in zip(self.weights,
nabla_w)]
            self.biases = [b-(eta/len(mini_batch))*nb
                           for b, nb in zip(self.biases,
nabla_b)]


        def backprop(self, x, y):
            """Повернути кортеж (nabla_b, nabla_w), що
представляє градієнт для функції вартості C_x. Nabla_b і nabla_w
- пошарові списки масивів numpy, схожі на self.biases і
self.weights. """
            nabla_b = [np.zeros(b.shape) for b in self.biases]
            nabla_w = [np.zeros(w.shape) for w in self.weights]
            # прямий прохід
            activation = x
            activations = [x] # список для пошарового зберігання
активацій
            zs = [] # список для пошарового зберігання z-векторів
            for b, w in zip(self.biases, self.weights):
                z = np.dot(w, activation)+b
                zs.append(z)
                activation = sigmoid(z)
                activations.append(activation)
            # зворотний прохід
            delta = self.cost_derivative(activations[-1], y) * \
                sigmoid_prime(zs[-1])
            nabla_b[-1] = delta
```

```
            nabla_w[-1] = np.dot(delta, activations[-
2].transpose())

            """Змінна l в циклі нижче використовується не так, як
описано вище. l = 1 означає останній шар нейронів, l = 2 -
передостанній, і так далі. Ми користуємося перевагою того, що в
python можна використовувати негативні індекси в масивах."""

        for l in xrange(2, self.num_layers):

                z = zs[-l]

                sp = sigmoid_prime(z)

                delta = np.dot(self.weights[-l+1].transpose(),
delta) * sp

                nabla_b[-l] = delta

                nabla_w[-l] = np.dot(delta, activations[-l-
1].transpose())

            return (nabla_b, nabla_w)


        def evaluate(self, test_data):

            """Повернути кількість перевірочних вхідних даних,
для яких нейронна мережа видає правильний результат. Вихідні дані
мережі - це номер нейрона в останньому шарі з найвищим рівнем
активації."""

            test_results = [(np.argmax(self.feedforward(x)), y)

                            for (x, y) in test_data]

            return sum(int(x == y) for (x, y) in test_results)


        def cost_derivative(self, output_activations, y):

            """Повернути вектор часткових похідних (partial C_x /

            \partial) для вихідних активацій."""

            return (output_activations-y)


    #### Різні функції

    def sigmoid(z):
```

```
        """Сгмоїда."""
        return 1.0/(1.0+np.exp(-z))


    def sigmoid_prime(z):
        """Похідна сигмоїди."""
        return sigmoid(z)*(1-sigmoid(z))
```

2. ПРАКТИЧНА ЧАСТИНА

Весь вихідний код і дані MNIST доступний у файлообміннику! Перейдемо в каталог з вихідним кодом та створимо віртуальне середовище та встановимо необхідні пакети:

```
roman@roman-l340:~/neural-networks$ virtualenv env
Running virtualenv with interpreter /usr/bin/python2
New python executable in /home/roman/neural-networks/env/bin/python2
Also creating executable in /home/roman/neural-networks/env/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
roman@roman-l340:~/neural-networks$ . env/bin/activate
(env) roman@roman-l340:~/neural-networks$ pip install -r requirements.txt
Collecting numpy==1.13.3
  Using cached numpy-1.13.3-cp27-cp27mu-manylinux1_x86_64.whl (16.6 MB)
Installing collected packages: numpy
Successfully installed numpy-1.13.3
(env) roman@roman-l340:~/neural-networks$ python --version
Python 2.7.17
```

Завантажимо дані MNIST. Зробимо це за допомогою невеликої допоміжної програми **mnist_loader.py**. Перейдемо в каталог **src.** Виконаємо такі команди в оболонці **python**:

```
(env) roman@roman-l340:~/neural-networks$ cd src/
(env) roman@roman-l340:~/neural-networks/src$ python
Python 2.7.17 (default, Feb 27 2021, 15:10:58)
[GCC 7.5.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import mnist_loader
>>> training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
```

```
    >>> import mnist_loader
    >>> training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
```

Після завантаження даних **MNIST** налаштуємо мережу з 30 прихованих нейронів. Це ми зробимо після імпорту описаної вище програми, яка називається **network**:

```
    >>> import network
    >>> net = network.Network([784, 30, 10])
```

Використовуємо стохастичний градієнтний спуск для навчання на навчальних даних протягом 30 епох, з розміром міні-пакета в 10, і швидкістю навчання η = 3.0 (виконання коду займає досить багато часу):

```
>>> net.SGD(training_data, 30, 10, 3.0, test_data=test_data)
```

Програма пише кількість правильно розпізнаних перевірочних зображень після кожної епохи тренувань. Як бачимо, навіть після однієї епохи вона досягає точності в 9 042 з 10 000, і це число продовжує зростати:

```
Epoch 0: 9042 / 10000
Epoch 1: 9224 / 10000
Epoch 2: 9278 / 10000
    ***
Epoch 27: 9526 / 10000
Epoch 28: 9517 / 10000
Epoch 29: 9501 / 10000
```

Виходить, що навчена мережа дає відсоток правильної класифікації близько 95% на максимумі. Досить багатообіцяюча перша спроба. Попереджаю, що у вас код не обов'язково буде видавати такі самі результати, оскільки ініціалізуємо мережу випадковими вагами і зміщеннями.

Перезапустимо експеримент, змінивши кількість прихованих нейронів до 100. Як і раніше, виконання коду займає досить багато часу (на моїй машині кожна епоха займає кілька десятків секунд):

```
>>> net = network.Network([784, 100, 10])

>>> net.SGD(training_data, 30, 10, 3.0, test_data=test_data)
```

Природно, це покращує результат до 96%. В цьому випадку використання більшої кількості нейронів допомагає отримувати кращі результати.

Звичайно, щоб досягти такої точності, потрібно було вибирати певну кількість епох для навчання, розмір міні-пакета і швидкість навчання η. Як згадувалося вище, їх називають гіперпараметрами нашої нейронної мережі - щоб відрізняти їх від простих параметрів (ваг і зміщень), які алгоритм налаштовує в процесі навчання. Якщо ми погано виберемо гіперпараметри, ми отримаємо погані результати. Припустимо, наприклад, що ми вибрали швидкість навчання η = 0,001:

```
>>> net = network.Network([784, 100, 10])

>>> net.SGD(training_data, 30, 10, 0.001,
test_data=test_data)
```

Результати вийдуть куди менш виражаючим:

```
Epoch 0: 1377 / 10000
Epoch 1: 1575 / 10000
Epoch 2: 1642 / 10000
      ***
Epoch 27: 2451 / 10000
Epoch 28: 2472 / 10000
Epoch 29: 2488 / 10000
```

Однак можна бачити, що ефективність мережі з часом повільно зростає. Це говорить про те, що можна спробувати збільшити швидкість навчання, припустимо, до 0,01. У цьому випадку результати будуть краще, що говорить про необхідність ще більше збільшити швидкість. Якщо зробити це кілька разів, ми в підсумку прийдемо до $\eta = 1,0$ (а іноді навіть і 3,0), що близько до наших раннім експериментом. Так що, хоча спочатку погано вибрали гіперпараметри, зібрали достатньо інформації, щоб зуміти поліпшити наш вибір параметрів.

В цілому, налагодження нейронної мережі - справа складна. Особливо це так, коли вибір початкових гіперпараметрів видає результати, які не перевищують випадкового шуму. Припустимо, що спробуємо використовувати успішну архітектуру з 30 нейронів, проте поміняємо швидкість навчання на 100,0:

```
>>> net = network.Network([784, 30, 10])

>>> net.SGD(training_data, 30, 10, 100.0,
test_data=test_data)
```

У підсумку виявиться, що зайшли занадто далеко, і взяли занадто велику швидкість:

```
Epoch 0: 1028 / 10000
Epoch 1: 1028 / 10000
Epoch 2: 1028 / 10000
      ***
Epoch 27: 1028 / 10000
Epoch 28: 1028 / 10000
Epoch 29: 1028 / 10000
```

Тепер уявіть, що підходимо до цього завдання в перший раз. Звичайно, знаємо з ранніх експериментів, що правильно буде зменшити швидкість навчання. Але якби ми підступалися до цього завдання вперше, у нас не було б вихідних даних, здатних привести нас до вірного рішення. Могли б припустити те, що, можливо, вибрали неправильні початкові параметри для ваг і зміщень, і мережі важко навчатися? Або, можливо, у нас недостатньо навчальних даних, щоб отримати осмислений результат? Можливо, ми не почекали досить епох? Можливо, нейронна мережа з такою архітектурою просто не може навчитися розпізнавати рукописні цифри? Можливо, швидкість навчання занадто мала? При першому підході до задачі у вас ніколи немає впевненості.

З цього варто винести урок про те, що налагодження нейронної мережі не є тривіальним завданням, і це, як і звичайне програмування, є частиною мистецтва. Ви повинні навчитися цьому мистецтву налагодження, щоб отримувати хороші результати від нейронної мережі.

3. ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Ознайомитись з теоретичними відомостями до лабораторної роботи.

2. Завантажити вихідний код з файлообмінника.

3. Провести навчання мережі з 30 прихованими нейронами, протягом 30 епох, з розміром міні-пакета в 10, і швидкістю навчання $\eta = 3.0$

4. Виконати індивідуальне завданням згідно варіанту.

5. Оформити звіт.

6. Зробити висновок згідно отриманих результатів.

Завдання

Варіант	Кількість прихованих нейронів	Кількість епох	Розмір міні-пакета	Швидкістю навчання
1	30	15	8	3.0
2	40	20	9	2.0
3	50	25	10	0.8
4	60	10	10	0.5
5	70	30	11	0.3
6	80	28	12	0.1
7	90	25	13	0.05
8	100	30	14	0.04
9	85	20	15	0.03
10	55	20	11	1.0

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Лупенко С. А. Теоретичні основи моделювання та опрацювання циклічних сигналів в інформаційних системах. – Львів: Магнолія - 2006, 2016. 344 с.

2. Лупенко С.А., Шаблій Н.Р. Конспект лекцій з дисципліни «Математичне забезпечення» комп'ютерних систем та мереж» для студентів денної та заочної форм навчання спеціальності 123 «Комп'ютерна інженерія». Тернопіль : ТНТУ, 2021. 210 с.

3. Кононюк А.Ю. Нейроні мережі і генетичні алгоритми. – К. : «Корнійчук». 2008. 446 с.

4. Томашевський В.М. Моделювання систем. – Київ: Видавнича група ВНУ, 2005. 352 с.

5. Вакарчук І.О. Квантова механіка. 4-е видання, доповнене. – Л.: ЛНУ ім. Івана Франка, 2012. 872 с.

6. Ткачук В.М. Фундаментальні проблеми квантової механіки. Л.: ЛНУ ім. Івана Франка, 2011. – 144 с.

7. Николайчук Я.М., Пітух І.Р., Возна Н.Я. Теорія моделей руху даних розподілених комп'ютерних систем. Монографія - Тернопіль: ТзОВ "Терног раф", 2008. 216 с.

8. Duckett J. JavaScript and jQuery: Interactive Front-End Web Development 1st Edition. 2022. p. 1370.

9. Теслюк В.М. Моделі та інформаційні технології синтезу мікроелектромеханічних систем: Монографія. Львів: Видавництво ПП "Вежа і Ко", 2018. 192 с.

10. Hanspeter Mössenböck, Johannes Kepler. The Compiler Generator Coco/R. User Manual.- University of Linz, 2006.

11. Lupenko S., Butsiy R., Volyanyk O., Stadnyk N. Advanced Signal Processing and Classification of EEG Patterns in Neurointerface Systems. ITTAP-2023: Information Technologies: Theoretical and Applied Problems. Ternopil Ivan Puluj National Technical University Ternopil, Ukraine, November 22-24, 2023. P.16.

12. Stadnyk N., Rokosh M., Pryimak M. Generative AI and its impact on labor productivity and the Global Econom. The 2nd International Workshop on Computer

Information Technologies in Industry 4.0 (CITI 2024). Ternopil, Ukraine, June 12-14, 2024. P. 175-186.

13. Galton A., Mizoguchi R. Formal Ontology in Information Systems (Frontiers in Artificial Intelligence and Applications). IOS Press. May 15, 2010. p. 440.

14. Вороновский Г.К., Махотило К.В., Петрашев С.Н., Сергєєв С.А. Генетичні алгоритми, штучні нейроні мережі і проблеми віртуальної реальності. Замовне. – Х.: ОСНОВА, 1997. С. 112.

15. Goodman D. JavaScript Bible. 7th edition. New York 2010. p. 1224.

16. Haykin S. Neural Networks: A Comprehensive Foundation Subsequent Edition. January 1, 1998. p. 842.

17. Николайчук Я.М., Возна Н.Я., Пітух І.Р. Проєктування спеціалізованих комп'ютерних систем. Навчальний посібник Тернопіль: ТзОВ "Терн граф". 2010. 392с.

18. Прокопенко Ю.В., Татарчук Д.Д., Казміренко В.А. Обчислювальна математика. К. НТУУ «КПІ», 2013. 224 с.

19. Кузьмін А.В., Кузьміна Н. М., Телейко А. Б. Символьні та наближені обчислення в системі Maple: Навч. посіб. МАУП, 2006–2008. Ч. 2. К. : ДП «Видавничий дім «Персонал», 2008. 128 с.

20. https://www.w3.org/Style/CSS/ CSS Home.

21. http://w3c.github.io/html/ HTML Editor's Draft.

22. https://www.cambridgesemantics.com/blog/semantic-university/learn-owl-rdfs/owl-101/ Вивчіть OWL і RDFS.

23. https://protegewiki.stanford.edu/wiki/Main_Page. Protégé.

24. The Advantages And Disadvantages Of Convolutional Neural Networks [Електронний ресурс] – Режим доступу до ресурсу: https://www.surfactants.net/the-advantages-and-disadvantages-of convolutional-neural-networks/

25. https://dl.tntu.edu.ua/bounce.php?course=1568 Електронні навчальні курси ТНТУ імені І. Пулюя.