ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ОЛЕСЯ ГОНЧАРА

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ ТА МАТЕМАТИЧНОЇ КІБЕРНЕТИКИ

ЗВІТ ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ № 2

З дисципліни «Сучасні середовища програмування»

Перший (бакалаврський) рівень вищої освіти

Спеціальність 113 Прикладна математика

Освітня програма Комп’ютерне моделювання

та технології програмування

Виконавець:

Студент(ка) групи ПА–21–1

Карпенко Владислав

Варіант № 6

Дніпро

2023

Зміст

[1. Постановка завдання 3](#_Toc163081991)

[2. Опис логічної структури Java-програми 3](#_Toc163081992)

[3. Результати роботи програми та деталі її застосування 4](#_Toc163081993)

[4. Висновки за результатами роботи 5](#_Toc163081994)

[5. Додатки (Код програми) 5](#_Toc163081995)

[Код основного класу Main.java 5](#_Toc163081996)

[Код класу Function.java 6](#_Toc163081997)

[Код класу Composite.java 6](#_Toc163081998)

[Код класу Const.java 7](#_Toc163081999)

[Код класу Linear.java 7](#_Toc163082000)

[Код класу Sum.java 8](#_Toc163082001)

[Код класу Multiplication.java 9](#_Toc163082002)

[Код класу Cosine.java 10](#_Toc163082003)

[Код класу Sine.java 10](#_Toc163082004)

[Код класу NaturalLogarithm.java 11](#_Toc163082005)

[Код класу Absolute.java 11](#_Toc163082006)

[Код класу HyperbolicCosine.java 12](#_Toc163082007)

[Код класу HyperbolicSine.java 12](#_Toc163082008)

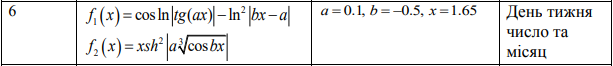
[Код класу Power.java 13](#_Toc163082009)

[Код класу Tangent.java 13](#_Toc163082010)

[Код класу Sign.java 14](#_Toc163082011)

1. Постановка завдання:

* Застосувати патерн Компонувальник (Composite) для задачі обчислення похідної довільної складеної функції. Використати дані таб. 1
* Індивідуальне завдання:



1. Опис логічної структури Java-програми:

Структура Java-програми складається з шістнадцяти класів: Main.java, Function.java, Composite.java, Const.java, Linear.java, Sum.java, Multiplication.java, Cosine.java, Sine.java, NaturalLogarithm.java, Absolute.java, HyperbolicCosine.java, HyperbolicSine.java, Power.java, Tangent.java та Sign.java.

Main.java – це основний клас, в якому створюються об’єкти класів функцій та виводяться результати роботи цих функцій.

Function.java – це інтерфейс, який наслідується до основних об’єктів типу функція.

Composite.java – це абстрактний клас, який імлементує Function, для обробки даних типу функція.

Const.java – це клас, який також імлементує Function, для роботи з числами, які потрібно ідентифікувати як константу.

Linear.java – це клас, який також імлементує Function, для роботи зі змінними, які надалі потрібно буде ідентифікувати, як лінійну функцію.

Sum.java – це клас, який наслідує Composite, цей клас слугує для знаходження суми результатів функцій які передані до цього класу як аргументи.

Multiplication.java – це клас, який також наслідує Composite, цей клас слугує для знаходження добутку результатів функцій які передані до цього класу як аргументи.

Cosine.java – це клас, який також наслідує Composite, цей клас слугує для знаходження значення косинусу від результатів функцій які передані до цього класу як аргументи.

Sine.java – це клас, який також наслідує Composite, цей клас слугує для знаходження значення синусу від результатів функцій які передані до цього класу як аргументи.

NaturalLogarithm.java – це клас, який також наслідує Composite, цей клас слугує для знаходження значення натурального логаріфму від результатів функцій які передані до цього класу як аргументи.

Absolute.java – це клас, який також наслідує Composite, цей клас слугує для модуля результатів функцій які передані до цього класу як аргументи.

HyperbolicCosine.java – це клас, який також наслідує Composite, цей клас слугує для знаходження значення гіперболічного косинусу від результатів функцій які передані до цього класу як аргументи.

HyperbolicSine.java – це клас, який також наслідує Composite, цей клас слугує для знаходження значення гіперболічного синусу від результатів функцій які передані до цього класу як аргументи.

Power.java – це клас, який також наслідує Composite, цей клас слугує для знаходження значення результату функції який передано до цього класу, як аргумент у степені, яку також передано, як аргумент.

Tangent.java – це клас, який також наслідує Composite, цей клас слугує для знаходження значення тангенсу від результатів функцій які передані до цього класу як аргументи.

Sign.java – це клас, який також наслідує Composite, цей клас слугує для повернення модулю результатів функцій які передані до цього класу як аргументи, об’єкт цього класу створюється коли програма знаходить похідну у класі Absolute.

1. Результати роботи програми та деталі її застосування:

Функція та похідна цієї функції:

f1 (x) = (cos(ln(|tg(0,1\*x)|))+(-1\*(ln(|(-0,5\*x-0,1)|))^(2)))

f1'(x) = ((-1\*sin(ln(|tg(0,1\*x)|))\*((|tg(0,1\*x)|)^(-1)\*(|tg(0,1\*x)|\*((cos(0,1\*x))^(-2)\*0,1))))+((0\*(ln(|(-0,5\*x-0,1)|))^(2))+(-1\*(2\*(ln(|(-0,5\*x-0,1)|))^((2-1))\*((|(-0,5\*x-0,1)|)^(-1)\*(|(-0,5\*x-0,1)|\*(-0,5+0)))))))

Значення функції та похідної:

f1 (1,65) = -0,226142

f1'(1,65) = 0,686354

Функція та похідна цієї функції:

f2(x) = (x\*(sh(|(0,1\*(cos(-0,5\*x))^(0,333))|))^(2))

f2'(x) = ((1\*(sh(|(0,1\*(cos(-0,5\*x))^(0,333))|))^(2))+(x\*(2\*(sh(|(0,1\*(cos(-0,5\*x))^(0,333))|))^((2-1))\*(ch(|(0,1\*(cos(-0,5\*x))^(0,333))|)\*(|(0,1\*(cos(-0,5\*x))^(0,333))|\*((0\*(cos(-0,5\*x))^(0,333))+(0,1\*(0,333\*(cos(-0,5\*x))^((0,333-1))\*(-1\*sin(-0,5\*x)\*-0,5)))))))))

Значення функції та похідної:

f2(1,65) = 0,012774

f2'(1,65) = 0,003121

Результати роботи (підтвердження правильності):

Значення першої функції при: a = 0.1; b = -0.5; x = 1.65

Изображение выглядит как текст, снимок экрана, веб-страница, Веб-сайт

Автоматически созданное описание

Значення похідної першої функції при: a = 0.1; b = -0.5; x = 1.65

Изображение выглядит как текст, снимок экрана, веб-страница, программное обеспечение

Автоматически созданное описание

Значення другої функції при: a = 0.1; b = -0.5; x = 1.65

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Значення похідної другої функції при: a = 0.1; b = -0.5; x = 1.65

Изображение выглядит как текст, снимок экрана, веб-страница, программное обеспечение

Автоматически созданное описание

1. Висновки за результатами роботи:

У цій лабораторній роботі я розробив програму, яка обчислює похідну функції та знаходить значення цієї функції у якійсь точці. Також у цій лабораторній роботі я познайомився з патерном Компанувальник, та використав його для розробки програми. У ході роботи програми помилок не виявлено. Програма працює правильно, похідні функцій та значення функції та похідної, які задані в індивідуальному завданні вирішує правильно.

1. Додатки (Код програми):

Код основного класу Main.java:

import java.text.NumberFormat;  
  
public class Main {  
 public static void main(String[] args) {  
 final double a = 0.1;  
 final double b = -0.5;  
 final double x0 = 1.65;  
  
 // f₁(x) = cos(ln|tg(ax)|) - ln²|bx-a|  
 // f₁'(x) = -sin(ln|tg(ax)|) / |tg(ax)| \* sign(tg(ax)) / cos²(ax) \* a  
 // - 2 \* ln|bx-a| / |bx-a| \* sign(bx-a) \* b  
 final Function expression1 =  
 Sum.*of*(  
 Cosine.*of*(  
 NaturalLogarithm.*of*(  
 Absolute.*of*(  
 Tangent.*of*(  
 Linear.*of*(a)  
 )  
 )  
 )  
 ),  
 Multiplication.*of*(  
 Const.*NEGATIVE\_ONE*,  
 Power.*of*(  
 NaturalLogarithm.*of*(  
 Absolute.*of*(  
 Sum.*of*(  
 Linear.*of*(b),  
 Const.*of*(-a)  
 )  
 )  
 ),  
 Const.*of*(2)  
 )  
 )  
 );  
 final NumberFormat nf = NumberFormat.*getInstance*();  
 System.*out*.format("f1 (x) = %s", expression1.toPrettyString(nf)).println();  
 System.*out*.format("f1'(x) = %s", expression1.derivative().toPrettyString(nf)).println();  
 System.*out*.format("f1 (%.2f) = %f", x0, expression1.calculate(x0)).println();  
 System.*out*.format("f1'(%.2f) = %f", x0, expression1.derivative().calculate(x0)).println();  
  
 // f₂(x) = xsh²|a∛(cos(bx))|  
 // f'₂(x) = sh²|a∛(cos(bx))|  
 // + x \* 2 \* sh|a∛(cos(bx))| \* ch|a∛(cos(bx))| \* sign(a∛(cos(bx))  
 // \* a \* (1/3) \* (cos(bx))^(-2/3) \* (-sin(bx)) \* b  
 final Function expression2 =  
 Multiplication.*of*(  
 Linear.*X*,  
 Power.*of*(  
 HyperbolicSine.*of*(  
 Absolute.*of*(  
 Multiplication.*of*(  
 Const.*of*(a),  
 Power.*of*(  
 Cosine.*of*(  
 Linear.*of*(b)  
 ),  
 Const.*of*(1.0 / 3.0)  
 )  
 )  
 )  
 ),  
 Const.*of*(2)  
 )  
 );  
 System.*out*.format("f2(x) = %s", expression2.toPrettyString(nf)).println();  
 System.*out*.format("f2'(x) = %s", expression2.derivative().toPrettyString(nf)).println();  
 System.*out*.format("f2(%.2f) = %f", x0, expression2.calculate(x0)).println();  
 System.*out*.format("f2'(%.2f) = %f", x0, expression2.derivative().calculate(x0)).println();  
 }  
}

Код класу Function.java:

import java.text.NumberFormat;  
  
public interface Function {  
 double calculate(double x);  
  
 Function derivative();  
  
 String toPrettyString(NumberFormat nf);  
}

Код класу Composite.java:

import java.util.ArrayList;  
import java.util.Arrays;  
  
public abstract class Composite implements Function {  
 private final ArrayList<Function> terms;  
  
 public ArrayList<Function> terms() {  
 return terms;  
 }  
  
 public Composite() {  
 terms = new ArrayList<>();  
 }  
  
 public Composite(Function... terms) {  
 this.terms = new ArrayList<>(Arrays.*asList*(terms));  
 }  
  
 public Composite(ArrayList<Function> terms) {  
 this.terms = terms;  
 }  
  
}

Код класу Const.java:

import java.text.NumberFormat;  
  
public class Const extends Number implements Function {  
 public static final Const *ZERO* = new Const(0);  
 public static final Const *ONE* = new Const(1);  
 public static final Const *NEGATIVE\_ONE* = new Const(-1);  
 private final double value;  
  
 public Const(double value) {  
 this.value = value;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return value;  
 }  
  
 @Override  
 public Function derivative() {  
 return *ZERO*;  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return nf.format(value);  
 }  
  
 @Override  
 public int intValue() {  
 return (int) value;  
 }  
  
 @Override  
 public long longValue() {  
 return (long) value;  
 }  
  
 @Override  
 public float floatValue() {  
 return (float) value;  
 }  
  
 @Override  
 public double doubleValue() {  
 return value;  
 }  
  
 public static Const of(double value) {  
 return new Const(value);  
 }  
}

Код класу Linear.java:

import java.text.NumberFormat;  
  
public class Linear implements Function {  
 public static final Linear *X* = new Linear(1.0) {  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return "x";  
 }  
 };  
 private final double coefficient;  
  
 public Linear(double coefficient) {  
 this.coefficient = coefficient;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return x \* coefficient;  
 }  
  
 @Override  
 public Function derivative() {  
 return new Const(coefficient);  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return String.*format*("%s\*x", nf.format(coefficient));  
 }  
  
  
 public static Linear of(double coefficient) {  
 return new Linear(coefficient);  
 }  
}

Код класу Sum.java:

import java.text.NumberFormat;  
import java.util.ArrayList;  
import java.util.StringJoiner;  
  
public class Sum extends Composite {  
 public Sum() {  
 super();  
 }  
  
 public Sum(Function... terms) {  
 super(terms);  
 }  
  
 public Sum(ArrayList<Function> terms) {  
 super(terms);  
 }  
  
 @Override  
 public double calculate(double x) {  
 double result = 0.0;  
 for (Function function : terms()) {  
 result += function.calculate(x);  
 }  
 return result;  
 }  
  
 @Override  
 public Function derivative() {  
 final ArrayList<Function> derivativeTerms = new  
 ArrayList<>(terms().size());  
 for (Function function : terms()) {  
 derivativeTerms.add(function.derivative());  
 }  
 return new Sum(derivativeTerms);  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 final StringJoiner joiner = new StringJoiner("+");  
 for (Function function : terms()) {  
 joiner.add(function.toPrettyString(nf));  
 }  
 return String.*format*("(%s)",  
 joiner.toString()).replace("+-", "-");  
 }  
  
 public static Sum of(Function... terms) {  
 return new Sum(terms);  
 }  
}

Код класу Multiplication.java:

import java.text.NumberFormat;  
import java.util.ArrayList;  
import java.util.StringJoiner;  
  
public class Multiplication extends Composite {  
 public Multiplication() {  
 super();  
 }  
  
 public Multiplication(Function... terms) {  
 super(terms);  
 }  
  
 public Multiplication(ArrayList<Function> terms) {  
 super(terms);  
 }  
  
 @Override  
 public double calculate(double x) {  
 double result = 1.0;  
 for (Function function : terms()) {  
 result \*= function.calculate(x);  
 }  
 return result;  
 }  
  
 @Override  
 public Function derivative() {  
 int size = terms().size();  
  
 final ArrayList<Function> derivativeTerms = new ArrayList<>(size);  
 for (Function function : terms()) {  
 derivativeTerms.add(function.derivative());  
 }  
  
 final ArrayList<Function> multiplicationTerms = new ArrayList<>(size);  
  
 for (int i = 0; i < size; i++) {  
 final ArrayList<Function> multiplicationTerms2 = new ArrayList<>(size);  
 for (int j = 0; j < size; j++) {  
 if (j == i) {  
 multiplicationTerms2.add(derivativeTerms.get(j));  
 } else {  
 multiplicationTerms2.add(terms().get(j));  
 }  
 }  
 multiplicationTerms.add(new Multiplication(multiplicationTerms2));  
 }  
  
 return new Sum(multiplicationTerms);  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 final StringJoiner joiner = new StringJoiner("\*");  
 for (Function function : terms()) {  
 joiner.add(function.toPrettyString(nf));  
 }  
 return String.*format*("(%s)", joiner.toString());  
 }  
  
 public static Multiplication of(Function... terms) {  
 return new Multiplication(terms);  
 }  
}

Код класу Cosine.java:

import java.text.NumberFormat;  
  
public class Cosine extends Composite {  
  
 public Cosine(Function term) {  
 super(term);  
 }  
  
 @Override  
 public double calculate(double x) {  
 return Math.*cos*(terms().getFirst().calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 Function term = terms().getFirst();  
 return new Multiplication(Const.*NEGATIVE\_ONE*, Sine.*of*(term), term.derivative());  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return *STR*."cos(\{terms().getFirst().toPrettyString(nf)})";  
 }  
  
 public static Cosine of(Function term){  
 return new Cosine(term);  
 }  
  
}

Код класу Sine.java:

import java.text.NumberFormat;  
  
public class Sine extends Composite {  
  
 public Sine(Function term) {  
 super(term);  
 }  
  
 @Override  
 public double calculate(double x) {  
 return Math.*sin*(terms().getFirst().calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 Function term = terms().getFirst();  
 return Multiplication.*of*(Cosine.*of*(term), term.derivative());  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return *STR*."sin(\{terms().getFirst().toPrettyString(nf)})";  
 }  
  
 public static Sine of(Function term) {  
 return new Sine(term);  
 }  
  
}

Код класу NaturalLogarithm.java:

import java.text.NumberFormat;  
  
public class NaturalLogarithm extends Composite {  
  
 public NaturalLogarithm(Function term) {  
 super(term);  
 }  
  
 @Override  
 public double calculate(double x) {  
 return Math.*log*(terms().getFirst().calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 Function term = terms().getFirst();  
 return new Multiplication(Power.*of*(term, Const.*NEGATIVE\_ONE*), term.derivative());  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return *STR*."ln(\{terms().getFirst().toPrettyString(nf)})";  
 }  
  
 public static NaturalLogarithm of(Function term){  
 return new NaturalLogarithm(term);  
 }  
  
}

Код класу Absolute.java:

import java.text.NumberFormat;  
  
public class Absolute extends Composite {  
  
 public Absolute(Function term) {  
 super(term);  
 }  
  
 @Override  
 public double calculate(double x) {  
 return Math.*abs*(terms().getFirst().calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 Function term = terms().getFirst();  
 return new Multiplication(Sign.*of*(term), term.derivative());  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return *STR*."|\{terms().getFirst().toPrettyString(nf)}|";  
 }  
  
 public static Absolute of(Function term){  
 return new Absolute(term);  
 }  
  
}

Код класу HyperbolicCosine.java:

import java.text.NumberFormat;  
  
public class HyperbolicCosine extends Composite {  
  
 public HyperbolicCosine(Function term) {  
 super(term);  
 }  
  
 @Override  
 public double calculate(double x) {  
 return Math.*cosh*(terms().getFirst().calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 Function term = terms().getFirst();  
 return Multiplication.*of*(HyperbolicSine.*of*(term), term.derivative());  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return *STR*."ch(\{terms().getFirst().toPrettyString(nf)})";  
 }  
  
 public static HyperbolicCosine of(Function term){  
 return new HyperbolicCosine(term);  
 }  
  
}

Код класу HyperbolicSine.java:

import java.text.NumberFormat;  
  
public class HyperbolicSine extends Composite {  
  
 public HyperbolicSine(Function term) {  
 super(term);  
 }  
  
 @Override  
 public double calculate(double x) {  
 return Math.*sinh*(terms().getFirst().calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 Function term = terms().getFirst();  
 return Multiplication.*of*(HyperbolicCosine.*of*(term), term.derivative());  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return *STR*."sh(\{terms().getFirst().toPrettyString(nf)})";  
 }  
  
 public static HyperbolicSine of(Function term){  
 return new HyperbolicSine(term);  
 }  
  
}

Код класу Power.java:

import java.text.NumberFormat;  
  
/\* f(x) = x ^ n \*/  
public class Power extends Composite {  
  
 public Power(Function base, Function exponent) {  
 super(base, exponent);  
 }  
  
 @Override  
 public double calculate(double x) {  
 Function base = terms().get(0);  
 Function exponent = terms().get(1);  
 return Math.*pow*(base.calculate(x), exponent.calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 Function base = terms().get(0);  
 Function exponent = terms().get(1);  
 return Multiplication.*of*(exponent, Power.*of*(base, Sum.*of*(exponent, Const.*NEGATIVE\_ONE*)), base.derivative());  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 Function base = terms().get(0);  
 Function exponent = terms().get(1);  
 return *STR*."(\{base.toPrettyString(nf)})^(\{exponent.toPrettyString(nf)})";  
 }  
  
 public static Power of(Function base, Function exponent) {  
 return new Power(base, exponent);  
 }  
}

Код класу Tangent.java:

import java.text.NumberFormat;  
  
public class Tangent extends Composite {  
  
 public Tangent(Function term) {  
 super(term);  
 }  
  
 @Override  
 public double calculate(double x) {  
 return Math.*tan*(terms().getFirst().calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 Function term = terms().getFirst();  
 return new Multiplication(Power.*of*(Cosine.*of*(term), Const.*of*(-2)), term.derivative());  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return *STR*."tg(\{terms().getFirst().toPrettyString(nf)})";  
 }  
  
 public static Tangent of(Function term){  
 return new Tangent(term);  
 }  
  
}

Код класу Sign.java:

import java.text.NumberFormat;  
  
public class Sign extends Composite {  
  
 public Sign(Function term) {  
 super(term);  
 }  
  
 @Override  
 public double calculate(double x) {  
 double termCalculateResult = terms().getFirst().calculate(x);  
 if (termCalculateResult > 0) {  
 return 1;  
 }  
 if (termCalculateResult < 0) {  
 return -1;  
 }  
 return 0;  
 }  
  
 @Override  
 public Function derivative() {  
 return Const.*ZERO*;  
 }  
  
 @Override  
 public String toPrettyString(NumberFormat nf) {  
 return *STR*."|\{terms().getFirst().toPrettyString(nf)}|";  
 }  
  
 public static Sign of(Function term){  
 return new Sign(term);  
 }  
  
}