|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| КНC-13 | РГР | MOEA Framework |  |  |
| Мальчишин Р. | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р. З. | |

Що таке MOEAFramework?

MOEAFramework є безкоштовною бібліотекою з відкритим кодом на Java для розробки і експериментування з багатокритеріальними еволюційними алгоритмами (MOEA) та іншими алгоритмами загального призначення. Крім цього в бібліотеку включено ще ряд інших алгоритмі, , в тому числі NSGA-II, NSGA-III, MOEA, GDE3 і МЕ / D. Крім того, МЕ Framework надає інструменти, необхідні для швидко проектувати, розробки, виконання і статистично тестування оптимізації. Наступні особливості MOEAFramework відрізняють його від наявних альтернатив:

* **Швидкі, надійні реалізації багатьох багатокритеріальних еволюційних алгоритмів.** MOEAFramework містить NSGA-II, NSGA-III, \_-MOEA, \_-NSGA-II,PAES, PESA2, SPEA2, IBEA, SMS-EMOA, GDE3, SMPSO, OMOPSO, CMA-ES, and MOEA/D. Ці алгоритми оптимізовані для роботи, що робить їх легко доступними для високопродуктивних програм. Він також підтримує бібліотеку JMetal і PISA, та забезпечує доступ до 30 алгоритмів багатокритеріальної оптимізації.
* **Розширюваний користувацькими алгоритмами, проблемами і операторами.** Framework надає базовий набір алгоритмів, тестові завдання та пошукові оператори, але також може бути легко розширений та включити додаткові компоненти. Використання Service Provider Interface (SPI), та проблеми нових алгоритмів легко інтегровані в MOEAFramework.
* **Модульна конструкція для побудови нових алгоритмів оптимізації з існуючих компонентів.** Добре структурований, об'єктно-орієнтоване проектування бібліотеки MOEAFramework дозволяє комбінувати існуючі компонент для побудови нових алгоритмів оптимізації. І в разі потреби функціональні можливості, що недоступні в рамках MOEAFramework, ви завжди можете розширити існуючий клас або додати нові класи для підтримки будь-якої необхідної функції.
* **Дозвільна ліцензія з відкритим вихідним кодом.** MOEAFramework ліцензований під вільним і відкритим GNU Lesser General Public License, версія 3 або (на Ваш розсуд) будь-якої пізнішої версії. Це дозволяє кінцевим користувачам навчатися, змінювати і поширювати MOEAFramework вільно.
* **Повністю документований вихідний код.** Вихідний код повністю документовані і часто оновлюється щоб залишатися у відповідності з будь-якими змінами. Крім того, нажається велика детальна інструкція з прикладами використання MOEAFramework.
* **Широка підтримка доступна в Інтернеті.** Як активно підтримуваний проект, фікси помилок і нові функції додаються постійно. Розробники постійно прагнуть покращити цей продукт. Для полегшення цього, офіційний сайт надає інструменти для повідомлень про помилки, запити на нові можливості, або отримання відповідей на запитання.
* **Більше 1200 тестів для забезпечення достовірності.** Кожен випуск MOEAFramework піддається всебічному тестуванню і контролю якості. І, якщо будуть виявлені які-небудь помилки, які залишаються в цьому тестуванні, вони оперативно видаляються і випускаються патчі.

Framework МЕ в даний час поширюється в трьох формах: 1) скомпільованих бінариних файлах; 2)вихідному коді; і 3) демонстраційному додатку.

**Компільовані файли.** Відкомпілювані бінарні файли містять повністю працюючий MOEA***.*** Всі необхідні сторонні бібліотеки, дані і документація надаються. Це завантаження рекомендується для розробників, інтегруючих MOEAFramework в існуючий проект.

**Вихідний код.** Дистрибутив вихідного коду містить всі вихідні коди, блок тестів, документація і файли даних. Цей розподіл дає користувачам повний контроль над MOEAFramework, тому будь-який компонент може буди модифікований при нобхідност. Таким чином, це завантаження рекомендується для розробників, бажаючих внести свій вклад або вивчити внутрішню роботу MOEAFramework.

**Демо-додаток.** Демо-додаток містить кілька інтерактивних демо-версії MOEAFramework і запускається подвійним клацанням на завантажений JAR файл. Цей файл призначений для користувача, щоб швидко дізнатися про MOEAFramework і його можливості.

Різні збірки MOEAFramework можна завантажити з сайту moeaframework.org/. Скомпільовані бінарні файли і вихідний коду упаковані в архіви (.tar.gz). Для користувачів Unix / Linux / Mac можна використати таку команду для розпакуваня архіву:

**tar -xzf MOEAFramework-2.12.tar.gz**

Користувачі Windows, повинні використовувати архіватор, утиліту, як 7-Zip, щоб витягти вміст архіву. 7-Zip є вільним, з відкритим вихідний кодом ,програму можна завантажити з http://www.7-zip.org/.

Установка залежностей

Програмні пакети, перераховані нижче, потрібні або рекомендуються для використання MOEAFramework. Будь-програмний пакет позначений як обов'язковий повинен бути встановлений на вашому комп'ютері, щоб використовувати MOEAFramework. Програмне забезпечення позначено як опція не повинна бути встановлена, але значно полегшить розробку.

Java 6 або пізніша версія, необхідна для будь-якої системи, що працює з MOEAFramework. Якщо завантажено скомпільовані бінарні файли або демо-додаток, вам потрібно тільки встановити Java Runtime Environment (JRE). Вихідний код вимагає наборі Java Development (JDK), що містить компілятор та інші інструменти розробника. Ми рекомендуємо один з наступних постачальників (більшість з яких безкоштовні):

Oracle - http://www.oracle.com/technetwork/java/javase/

* Для Windows, Linux і Solaris

JRockit JDK - http://www.oracle.com/technetwork/middleware/jrockit/

* Для Windows, Linux і Solaris
* Може забезпечити кращу продуктивність і масштабованість на Intel 32 і 64-розрядних архітектур

OpenJDK - http://openjdk.java.net/

* Для Ubuntu 8.04 (або більш пізньої версії), Fedora 9 (або новіша версія), Red Hat Enterprise Linux 5 OpenSUSE Debian GNU / Linux 5.0 і OpenSolaris

IBM - http://www.ibm.com/developerworks/java/jdk/

* Для AIX, Linux і г / OS

Apple - http://support.apple.com/kb/DL1572

Eclipse, і NetBeans два середовища розробки для написання, налагодження, тестування і запуску Java програми. Eclipse можна безкоштовно завантажити з eclipse.org/ для, а NetBeans може бути отриманий з netbeans.org/.Apache Ant є інструментом Java для автоматичного збору та упаковки проектів, схожий Make utility на Unix / Linux. Особам, що працюють з вихідним кодом слід розглянути установку Apache Ant, оскільки він дозволяє автоматизувати створення і тестування MOEAFramework. Ant можна завантажити з http://ant.apache.org/. Інструкціям по установці, передбаченим за допомогою Ant необхідно слідувати. Зверніть увагу, що Eclipse, містить Ant, так що немає необхідності встановлювати Eclipse, і Ant разом. При роботі з розподілом вихідного коду, необхідно правильно сконфігурувати Java середовище, щоб гарантувати, що всі ресурси доступні. Для цього, необхідно весь вихідний код імпортувати у Eclipse

Починаючи з версії 2.4, МЕ Framework і його залежностей можуть бути вирішені за допомогою Maven Система управління залежностями. Maven доступна з http://maven.apache.org/. Для роботи з MOEAFramework для вашого Maven проекту, додайте наступну залежність:

<dependency>

<groupId>org.moeaframework</groupId>

<artifactId>moeaframework</artifactId>

<version>2.12</version>

</dependency>

Маючи установку готового MOEAFramework і його залежностей, корисно запустити MOEA Diagnostic Tool для тестування, якщо установка була виконана успішно. Якщо діагностичний прилад з'являється і ви може запустити будь-який алгоритм, то установка пройшла успішно.

**Компільовані бінарні файли**. Необхідно запустити launch-diagnostic-tool.bat на Windows. Ви можете вручну запустити діагностичний інструмент за допомогою наступної команди:

java -Djava.ext.dirs=lib

org.moeaframework.analysis.diagnostics.LaunchDiagnosticTool

**Вихідний код.** Всередині Eclipse, перейдіть в src org moeaframework analysis diagnostic package у вікні Package Explorer. Клацніть правою кнопкою миші на LaunchDiagnosticTool.java і виберіть запуск.

**Демонстрація програми.** Двічі клацніть завантажений JAR-файлу. Якщо демонстраційне вікно не з'являється, спробуйте вручну запустити інструмент з за допомогою наступної команди:

Java -jar MOEAFramework-2,12-Demo.jar

Кілька прикладів застосування забезпечуються MOEAFramework в папці

MOEA Framework-2,12 / examples /. При використанні Eclipse, або Netbeans, ви можете запустити ці приклади безпосередньо в графічному інтерфейсі користувача. Наприклад, в Eclipse, клацніть правою кнопкою миші на Example1.java виберіть запуск. Для запуску з командного рядка, ви спочатку повинні скомпілювати код за допомогою компілятора Java (Javac), а потім, запустивши приклад в JVM (Java). Наприклад, ви можете скомпілювати і запустити Перший приклад на Windows, наступним чином:

javac -cp ".;examples;lib/\*" examples/Example1.java

java -cp ".;examples;lib/\*" Example1

При використанні Unix / Linux, необхідно замінити крапку з комою (;) на двокрапку (:) в наведених вище командах.

**Приклади**

MOEAFramework дозволяє спростити процес вирішення задач багатокритеріальної оптимізації з використанням багатокритеріальних еволюційних алгоритмів (MOEAs). Клас **Executor** надає всі необхідні можливості для виконання алгоритму оптимізації на конкретному завданні. Наприклад, цей приклад показує алгоритм NSGA-II на тестовій задачі UF1.

Клас Executor є відправною точкою при використанні MOEAFramework. Він забезпечує всі функції, необхідні для налаштування алгоритму оптимізації рішення задачі і отримує доступ до результатів. Результат виконання програми показує кілька рядків, що містять числа. Оскільки проблема UF1 складається з двох об'єктивних проблем, результат містить два стовпці з рішеннями. Кожен рядок представляє собою оптимальне рішення Парето для цього завдання.

Код програми:

import org.moeaframework.Executor;

import org.moeaframework.core.NondominatedPopulation;

import org.moeaframework.core.Solution;

/\*\*

\* Demonstrates using an Executor to solve the UF1 test problem with NSGA-II,

\* one of the most widely-used multiobjective evolutionary algorithms.

\*/

public class Example1 {

public static void main(String[] args) {

//configure and run this experiment

NondominatedPopulation result = new Executor()

.withProblem("UF1")

.withAlgorithm("NSGAII")

.withMaxEvaluations(10000)

.run();

//display the results

System.out.format("Objective1 Objective2%n");

for (Solution solution : result) {

System.out.format("%.4f %.4f%n",

solution.getObjective(0),

solution.getObjective(1));

}

}

}

Результати виконання:

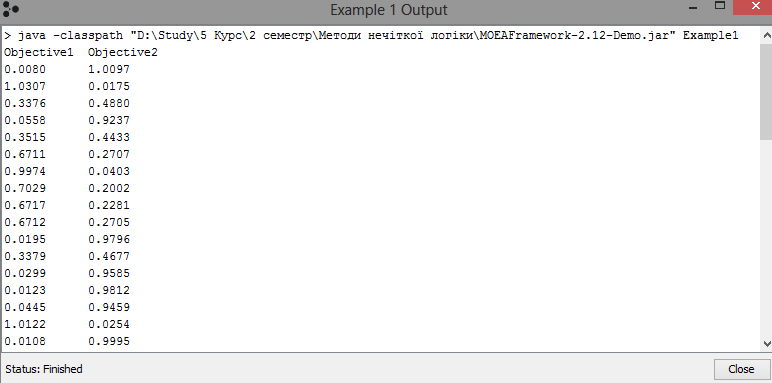


Рис. 1. Результат виконання Example1.java

MOEAFramework також надає безліч класів для виконання надійних статистичних аналізів для порівняння двох або більше алгоритмів оптимізації. Статистичний аналіз забезпечується класом Analyzer. Аналізатор може відображати хв, медіану, макс. і сумарних значення для безлічі показників ефективності, в тому числі відстані поколінь, обернуту відстань поколінь, адитивний індикатор, інтервал і вкладку. Крім того тести дають статистичні результати значущості.

Динамічне середовища виконання дає уявлення про поведінку алгоритму оптимізації протягом пробігу. Наприклад, можна спостерігати, як якість розчину змінюється з числом оцінок функції. Клас Instrumenter записує динаміку виконання.

Ряд методів доступні для надання користувацьких змін, які акуратно інтегруються з усіма іншими компонентами MOEAFramework. Цей приклад демонструє, як створити та оптимізувати двj-об'єктивну проблему DTLZ2. Example4.java показує, як ми створюємо клас з ім'ям MyDTLZ2 і визначаємо три методи:

* constructor,
* newSolution
* evaluate.

Усередині конструктора, ми просто викликаємо super (11, 2); інформуємо батьківський клас (AbstractProblem), що ця проблема має 11 змінних рішення і 2 завдання. Метод newSolution визначає уявлення завдання (кількість і тип її змінних рішення). Evaluate метод приймає рішення і обчислює значення цільової функції. Після того, як проблема визначена, вона може бути використана без проблем в MOEAFramework.

Код програми:

import org.moeaframework.Executor;

import org.moeaframework.core.NondominatedPopulation;

import org.moeaframework.core.Solution;

import org.moeaframework.core.variable.EncodingUtils;

import org.moeaframework.core.variable.RealVariable;

import org.moeaframework.problem.AbstractProblem;

/\*\*

\* Demonstrates how a new problem is defined and used within the MOEA

\* Framework.

\*/

public class Example4 {

/\*\*

\* Implementation of the DTLZ2 function.

\*/

public static class MyDTLZ2 extends AbstractProblem {

/\*\*

\* Constructs a new instance of the DTLZ2 function, defining it

\* to include 11 decision variables and 2 objectives.

\*/

public MyDTLZ2() {

super(11, 2);

}

/\*\*

\* Constructs a new solution and defines the bounds of the decision

\* variables.

\*/

@Override

public Solution newSolution() {

Solution solution = new Solution(getNumberOfVariables(),

getNumberOfObjectives());

for (int i = 0; i < getNumberOfVariables(); i++) {

solution.setVariable(i, new RealVariable(0.0, 1.0));

}

return solution;

}

/\*\*

\* Extracts the decision variables from the solution, evaluates the

\* Rosenbrock function, and saves the resulting objective value back to

\* the solution.

\*/

@Override

public void evaluate(Solution solution) {

double[] x = EncodingUtils.getReal(solution);

double[] f = new double[numberOfObjectives];

int k = numberOfVariables - numberOfObjectives + 1;

double g = 0.0;

for (int i = numberOfVariables - k; i < numberOfVariables; i++) {

g += Math.pow(x[i] - 0.5, 2.0);

}

for (int i = 0; i < numberOfObjectives; i++) {

f[i] = 1.0 + g;

for (int j = 0; j < numberOfObjectives - i - 1; j++) {

f[i] \*= Math.cos(0.5 \* Math.PI \* x[j]);

}

if (i != 0) {

f[i] \*= Math.sin(0.5 \* Math.PI \* x[numberOfObjectives - i - 1]);

}

}

solution.setObjectives(f);

}

}

public static void main(String[] args) {

//configure and run the DTLZ2 function

NondominatedPopulation result = new Executor()

.withProblemClass(MyDTLZ2.class)

.withAlgorithm("NSGAII")

.withMaxEvaluations(10000)

.run();

//display the results

System.out.format("Objective1 Objective2%n");

for (Solution solution : result) {

System.out.format("%.4f %.4f%n",

solution.getObjective(0),

solution.getObjective(1));

} }}

Результат виконання:

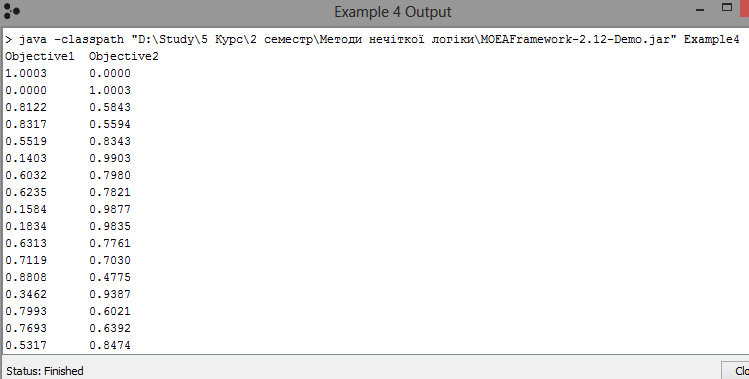


Рис. 2. Результат виконання Example4.java

Символьна регресія намагається виявити математичний вираз, який найкращим чином відповідає кривій. У цьому прикладі ми будемо намагатися знайти рівняння, щоб імітувати функцію F (X) = х ^ 4 + х ^ 3 + х ^ 2 + х.

Код програми:

package org.moeaframework.examples.gp.regression;

import org.apache.commons.math3.analysis.UnivariateFunction;

import org.moeaframework.core.Solution;

import org.moeaframework.core.variable.Program;

import org.moeaframework.problem.AbstractProblem;

import org.moeaframework.util.tree.Add;

import org.moeaframework.util.tree.Cos;

import org.moeaframework.util.tree.Divide;

/\*\*

\* The symbolic regression problem for genetic programming. Given a function,

\* the symbolic regression problem attempts to find an expression for closely

\* approximating the output of the function.

\*/

public class SymbolicRegression extends AbstractProblem {

/\*\*

\* The actual function implementation.

\*/

private final UnivariateFunction function;

/\*\*

\* The lower bound for comparing the actual and approximate functions.

\*/

private final double lowerBound;

/\*\*

\* The upper bound for comparing the actual and approximate functions.

\*/

private final double upperBound;

/\*\*

\* The number of comparisons made between the actual and approximate

\* functions.

\*/

private final int steps;

/\*\*

\* The name of the input variable for the expression tree.

\*/

private String symbol;

/\*\*

\* The rules for building expression trees for symbolic regression.

\*/

private Rules rules;

/\*\*

\* The cached x values.

\*/

private double[] x;

/\*\*

\* The cached actual y values.

\*/

private double[] y;

/\*\*

\* Constructs a new symbolic regression problem for approximating the

\* given function.

\*

\* @param function the actual function implementation

\* @param lowerBound the lower bound for comparing the actual and

\* approximate functions

\* @param upperBound the upper bound for comparing the actual and

\* approximate functions

\* @param steps the number of comparisons made between the actual and

\* approximate functions

\*/

public SymbolicRegression(UnivariateFunction function,

double lowerBound, double upperBound, int steps) {

super(1, 1);

this.function = function;

this.lowerBound = lowerBound;

this.upperBound = upperBound;

this.steps = steps;

// setup the default rules

symbol = "x";

rules = new Rules();

rules.add(new Add());

rules.add(new Multiply());

rules.add(new Subtract());

rules.add(new Divide());

rules.add(new Sin());

rules.add(new Cos());

rules.add(new Exp());

rules.add(new Log());

rules.add(new Get(Number.class, symbol));

rules.setReturnType(Number.class);

rules.setMaxVariationDepth(10);

// cache the function's x and y values

x = new double[steps];

y = new double[steps];

for (int i = 0; i < steps; i++) {

x[i] = lowerBound + (i / (steps-1.0)) \* (upperBound - lowerBound);

y[i] = function.value(x[i]);

}}

/\*\*

\* Returns the actual function implementation.

\*

\* @return the actual function implementation

\*/

public UnivariateFunction getFunction() {

return function;

}

@Override

public Solution newSolution() {

Solution solution = new Solution(1, 1);

solution.setVariable(0, new Program(rules));

return solution;}}

Результат виконання:

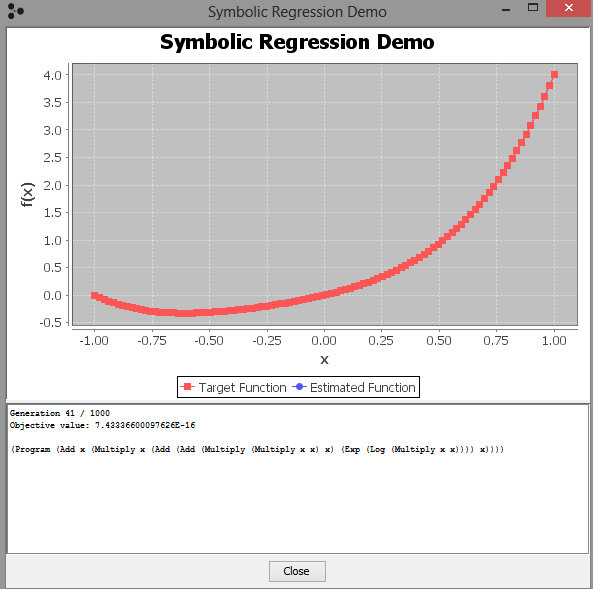


Рис. 3. Результат виконання SymbolicRegression.java

**Запуск в Intellij IDEA**

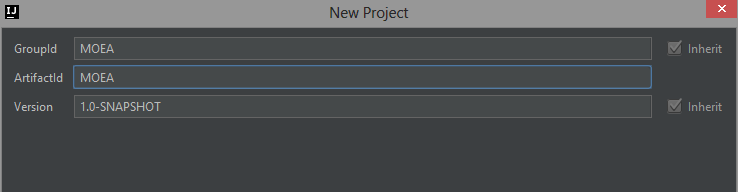


Рис. 4. Створення maven проекту.

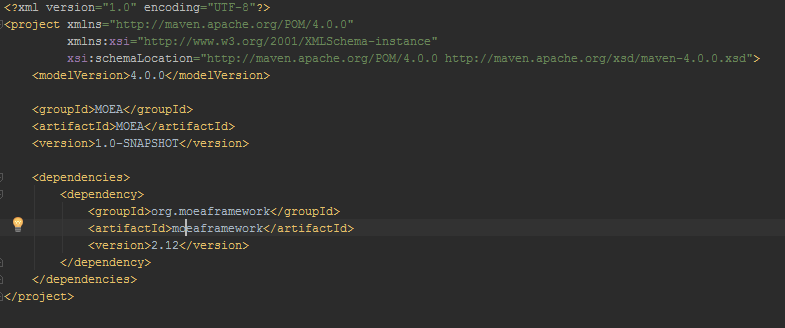


Рис. 5. Додавання залежності на MOEA Framework.

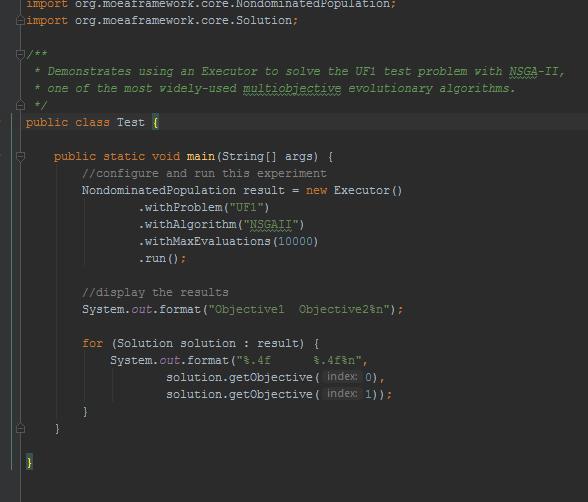


Рис. 6. Створення тестового класу.

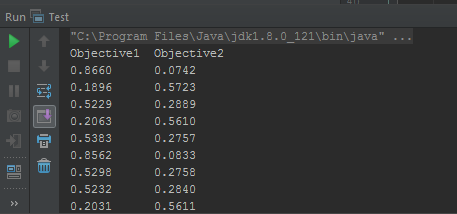


Рис. 7. Результат виконання Test.java

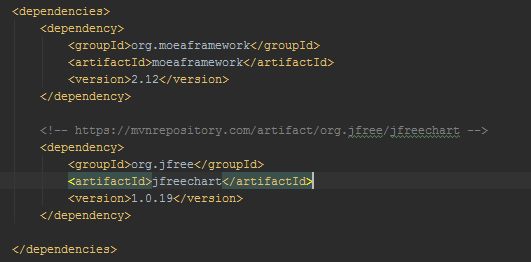


Рис. 8. Додавання залежності на JFreeChart.

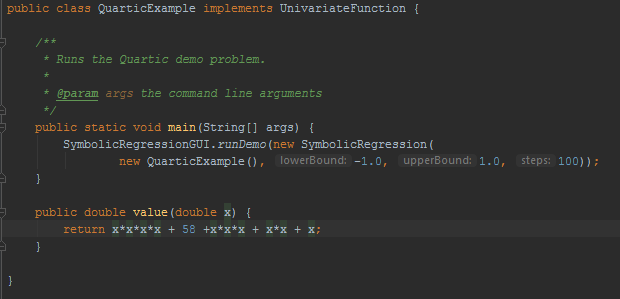


Рис. 9. Створення тестового класу.

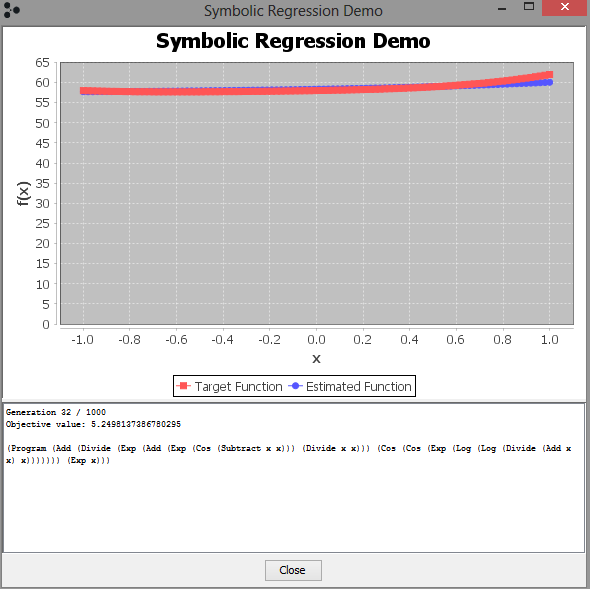


Рис. 10. Результат виконання Test.java