Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Ижевский государственный технический университет имени М.Т. Калашникова»

Кафедра «Программное обеспечение»

Отчет

по лабораторной работе №2 по теме

«Критерии. Критериальные функции. Поиск минимума или максимума»

по дисциплине

«Исследование операций. Теория принятия решений»

|  |  |
| --- | --- |
| Выполнил:  студент гр. Б06-191-1 | Мусин Р.И. |
| Принял: | Лугачев П.П. |

Ижевск 2018

ПОСТАНОВКА ЗАДАЧИ

Необходимо найти точку экстремума для функции от двух переменных методом регулярного симплекса. Написать программы на языке высокого уровня для нахождения минимума функции f(x, y) = 350x2 + 525у2 + 600 методом регулярного симплекса.

КОД ПРОГРАММЫ

**package** rustam.omk;  
  
**import** javafx.application.Application;  
**import** javafx.beans.binding.Bindings;  
**import** javafx.geometry.Insets;  
**import** javafx.geometry.Side;  
**import** javafx.scene.Node;  
**import** javafx.scene.Scene;  
**import** javafx.scene.chart.NumberAxis;  
**import** javafx.scene.layout.Pane;  
**import** javafx.scene.layout.StackPane;  
**import** javafx.scene.paint.Color;  
**import** javafx.scene.shape.Circle;  
**import** javafx.scene.shape.Line;  
**import** javafx.stage.Stage;  
  
**import** java.util.Arrays;  
**import** java.util.Comparator;  
  
**import static** java.lang.Math.*sqrt*;  
  
*// Java 8 code***public class** MainController **extends** Application {  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 **public void** start(**final** Stage stage) {  
 Axes axes = **new** Axes(  
 800, 800,  
 -10, 10, 1,  
 -10, 10, 1  
 );  
  
 Point[] startPoints = {  
 **new** Point(8, -8),  
 **new** Point(5, -8),  
 **new** Point(6.5, -8 + 3\*(*sqrt*(3)/2))  
 };  
 Algorithm algorithm = **new** Algorithm(startPoints);  
 Node[] allNodes = **new** Node[101];  
 allNodes[0] = **new** MyCircle(0, 0, 5, axes);  
 **for** (**int** i = 1; i < allNodes.**length**; i++) {  
 allNodes[i] = **new** MyTriangle(axes, algorithm.**all**);  
 algorithm.makeIteration();  
 }  
  
 StackPane layout = **new** StackPane(  
 allNodes  
 );  
 layout.setPadding(**new** Insets(20));  
 layout.setStyle(**"-fx-background-color: rgb(35, 39, 50);"**);  
  
 stage.setTitle(**"Привет"**);  
 stage.setScene(**new** Scene(layout, Color.*rgb*(35, 39, 50)));  
 stage.show();  
 }  
  
 **class** Axes **extends** Pane {  
 **private** NumberAxis **xAxis**;  
 **private** NumberAxis **yAxis**;  
  
 **public** Axes(  
 **int** width, **int** height,  
 **double** xLow, **double** xHi, **double** xTickUnit,  
 **double** yLow, **double** yHi, **double** yTickUnit  
 ) {  
 setMinSize(Pane.***USE\_PREF\_SIZE***, Pane.***USE\_PREF\_SIZE***);  
 setPrefSize(width, height);  
 setMaxSize(Pane.***USE\_PREF\_SIZE***, Pane.***USE\_PREF\_SIZE***);  
  
 **xAxis** = **new** NumberAxis(xLow, xHi, xTickUnit);  
 **xAxis**.setSide(Side.***BOTTOM***);  
 **xAxis**.setMinorTickVisible(**false**);  
 **xAxis**.setPrefWidth(width);  
 **xAxis**.setLayoutY(height / 2);  
  
 **yAxis** = **new** NumberAxis(yLow, yHi, yTickUnit);  
 **yAxis**.setSide(Side.***LEFT***);  
 **yAxis**.setMinorTickVisible(**false**);  
 **yAxis**.setPrefHeight(height);  
 **yAxis**.layoutXProperty().bind(  
 Bindings.*subtract*(  
 (width / 2) + 1,  
 **yAxis**.widthProperty()  
 )  
 );  
  
 getChildren().setAll(**xAxis**, **yAxis**);  
 }  
  
 **public** NumberAxis getXAxis() {  
 **return xAxis**;  
 }  
  
 **public** NumberAxis getYAxis() {  
 **return yAxis**;  
 }  
  
 **double** mapX(**double** x) {  
 **double** tx = getPrefWidth() / 2;  
 **double** dx = getXAxis().getUpperBound() - getXAxis().getLowerBound();  
 **double** sx = getPrefWidth() / dx;  
 **return** x \* sx + tx;  
 }  
  
 **double** mapY(**double** y) {  
 **double** ty = getPrefHeight() / 2;  
 **double** dy = getYAxis().getUpperBound() - getYAxis().getLowerBound();  
 **double** sy = getPrefHeight() / dy;  
 **return** -y \* sy + ty;  
 }  
 }  
  
 **class** MyTriangle **extends** Pane {  
 MyTriangle(Axes axes, Point[] all) {  
 **for** (**int** i = 0; i < all.**length**; i++) {  
 Point first = all[i];  
 **double** x0 = axes.mapX(first.**x**);  
 **double** y0 = axes.mapY(first.**y**);  
 **for** (**int** j = i + 1; j < all.**length**; j++) {  
 Point second = all[j];  
 **double** x1 = axes.mapX(second.**x**);  
 **double** y1 = axes.mapY(second.**y**);  
 Line line = **new** Line(x0, y0, x1, y1);  
 line.setStrokeWidth(.5f);  
 line.setStroke(Color.***GREEN***);  
 getChildren().add(line);  
 }  
 }  
 }  
 }  
  
 **class** MyCircle **extends** Pane {  
 **public** MyCircle(**double** x0, **double** y0, **double** radius, Axes axes) {  
 **double** xLeft = axes.mapX(x0 - radius);  
 **double** xRight = axes.mapX(x0 + radius);  
 **double** centerX = (xLeft + xRight) / 2;  
  
 **double** yTop = axes.mapY(y0 - radius);  
 **double** yBot = axes.mapY(y0 + radius);  
 **double** centerY = (yTop + yBot) / 2;  
  
 **double** r = xRight - centerX;  
  
 Circle circle = **new** Circle(centerX, centerY, r);  
 circle.setFill(Color.***TRANSPARENT***);  
 circle.setStroke(Color.***RED***);  
  
 getChildren().setAll(axes, circle);  
 }  
 }  
  
 **class** Algorithm {  
  
 Point[] **all**;  
  
 Algorithm(Point[] all) {  
 **this**.**all** = all;  
 }  
  
 **void** makeIteration() {  
 Arrays.*sort*(**all**, Comparator.*comparingDouble*(**this**::f));  
 **for** (**int** i = 2; i >= 0; i--) {  
 Point p = **all**[i].clone();  
 **double** prev = f(p);  
 **for** (**int** j = 0; j < 3; j++) {  
 **double** dx = **all**[j].**x** - **all**[i].**x**;  
 **double** dy = **all**[j].**y** - **all**[i].**y**;  
 p.**x** += dx;  
 p.**y** += dy;  
 }  
 **double** cur = f(p);  
 **if** (cur < prev) {  
 **all**[i] = p;  
 **return**;  
 }  
 }  
 decreaseTriangleSize();  
 }  
  
 **void** decreaseTriangleSize() {  
 Point p = **all**[0];  
 Point[] next = **new** Point[3];  
 **for** (**int** i = 0; i < 3; i++) {  
 **double** dx = **all**[i].**x** - p.**x**;  
 **double** dy = **all**[i].**y** - p.**y**;  
 next[i] = **new** Point(p.**x** + dx / 2, p.**y** + dy / 2);  
 }  
 **all** = next;  
 }  
  
 **double** f(Point p) {  
 **double** x = p.**x**;  
 **double** y = p.**y**;  
 **return** 350 \* x \* x + 525 \* y \* y + 600;  
 }  
 }  
}

РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

Результаты работы программы представлены на рисунке 1.

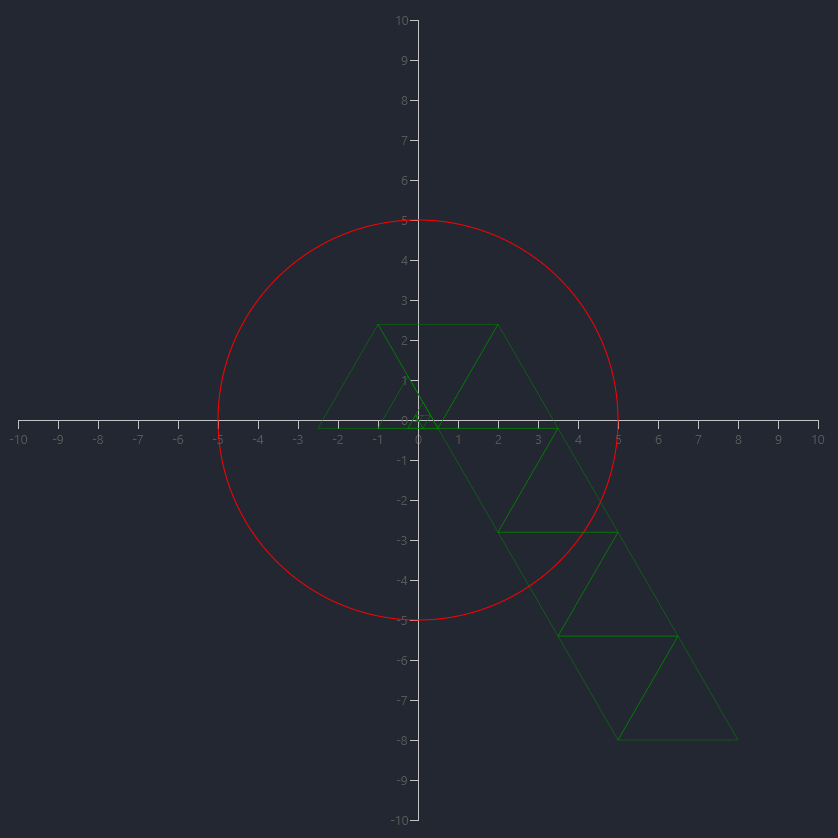


Рис.1. Результаты работы программы

ВЫВОД

В ходе выполнения лабораторной работы был изучен метод регулярного симплекса, решена задача по поиску минимума функции от двух аргументов.