группа №в3530904-10022

**17 вариант. 1 метод:**

**«Реализация метода случайного поиска для поиска минимума функции»**

1. **Описание общего алгоритма для метода случайного поиска:**
2. Задаем исходные данные:

Инициализируем параметры:

- начальная точка поиска A0(x,y);

- размер шага поиска –h;

- точность метода – epsilon;

- задаем границы области поиска и количество итераций. (не обязательно, в зависимости от реализации)

1. Направление поиска выбирается случайным образом.
2. Шаг в выбранном направлении дает «пробную точку».
3. Проверка пробной точки: Расчет значения функции в новой точке и сравнение значения функции в новой точке с лучшим значением, найденным на предыдущих итерациях.

Если значение функции в новой точке меньше лучшего значения, то она становится лучшей точкой, и ее координаты и значение функции заменяют лучшие координаты и значение функции.

1. Завершение: Комбинация условий:

- Если разница между текущим значением функции и лучшим значением меньше порога сходимости (epsilon), то поиск завершается. Найденная точка является минимумом функции.

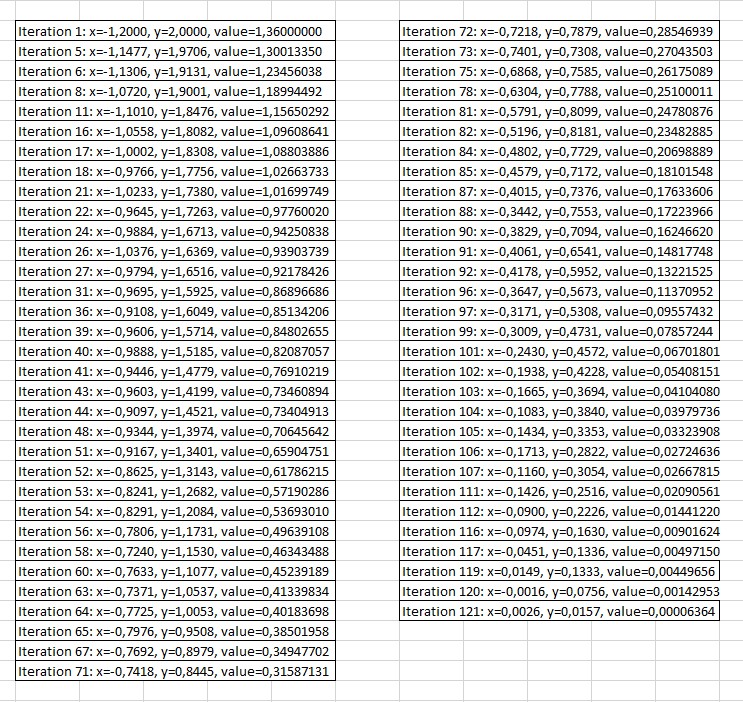
- Если все итерации были выполнены и лучшее значение функции не улучшилось, то завершаем поиск**. K > Kmax**

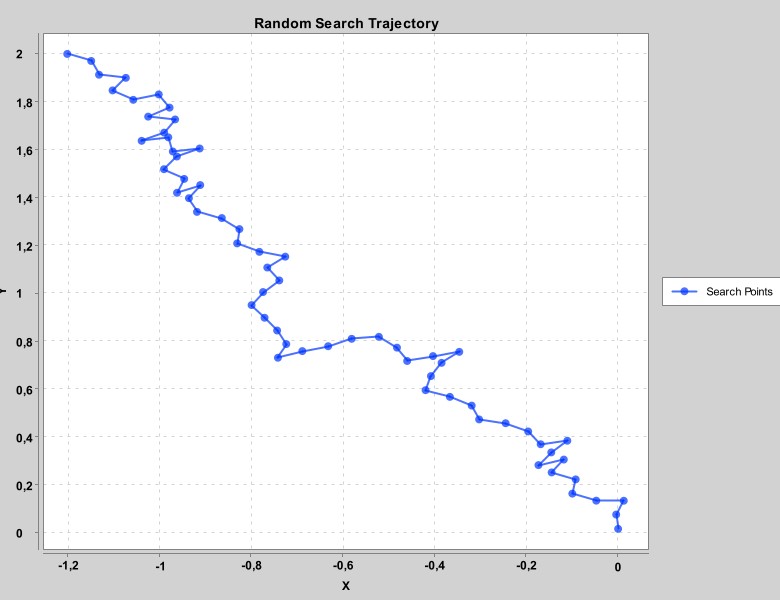
6) Повторять шаги 2, 3, 4 и проверять условия окончания поиска.

1. **Тестирование на функции эллипсоида:**

**Функция эллипсоида:**

**Параметры поиска:** A0(-1.2, 2), h = 0.06 epsilon = 0.0001, A=2, B = 2

**Таблица:**

**График траектории поиска:**

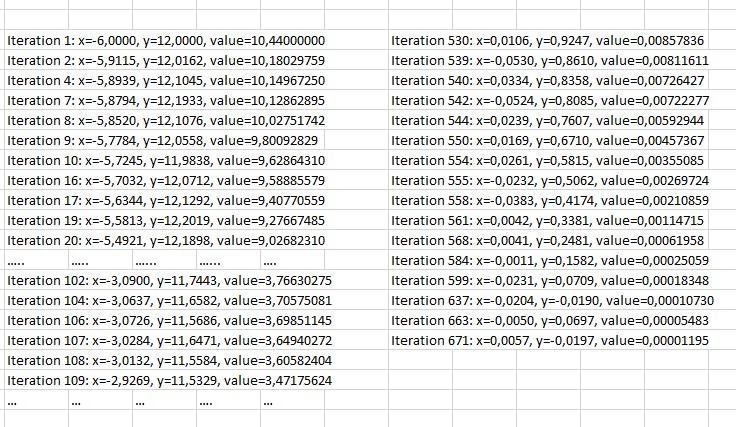
**Результаты исследования:**

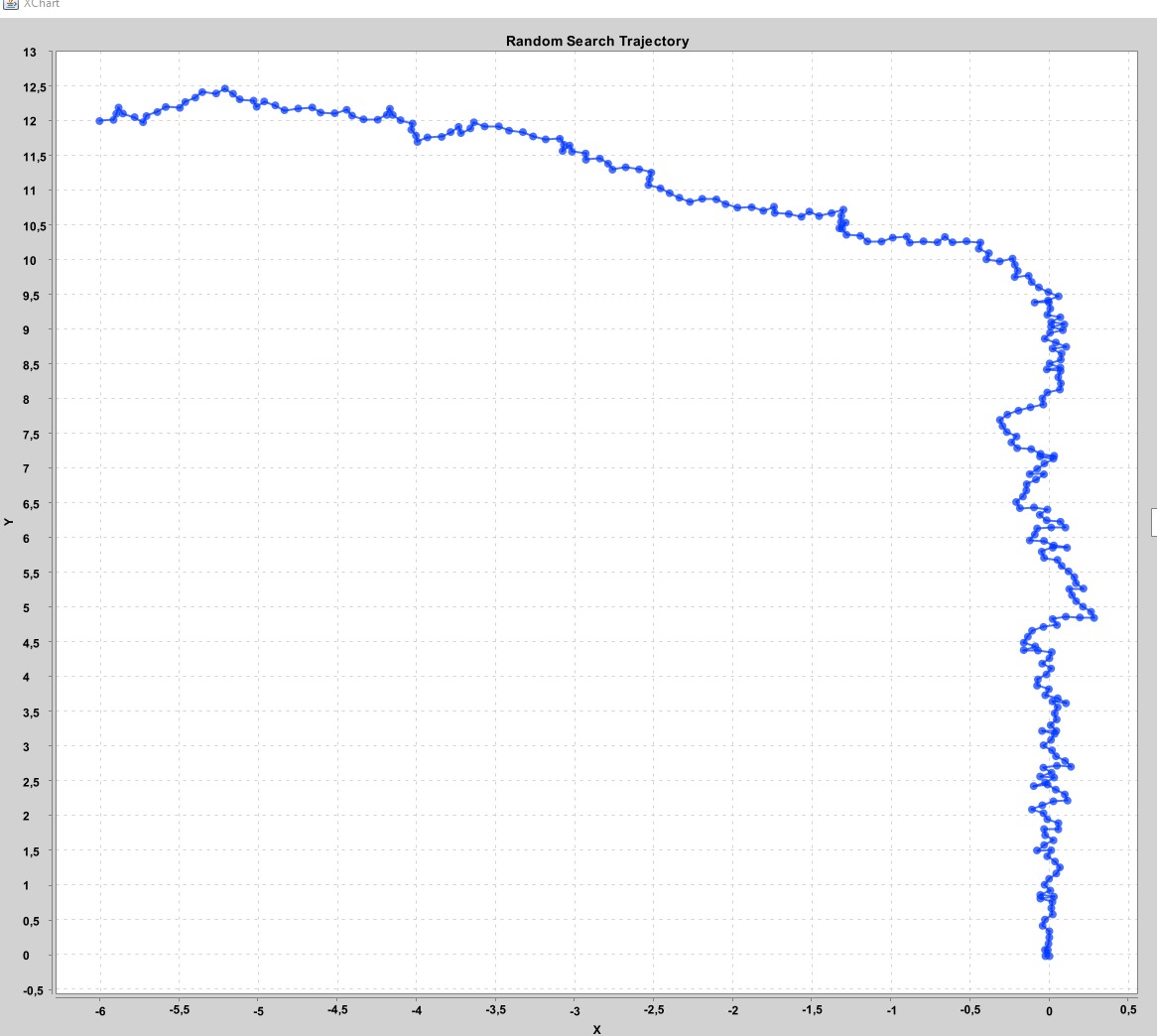
При значениях шага 0.06, радиусов эллипсоида А = 2 B = 2, координаты начальной точки A0(-1.2, 2) – метод случайного поиска достиг минимального значения функции в 0,00006364 в точках x=0,0026 y = 0,0157 за 121 итераций.

Поскольку метод случайного поиска выбирает направление случайно – с помощью формулы random – в зависимости от выбора начальной точки, шага, точности и параметров радиуса можно получить разные результаты.

Тест 2.

**Параметры поиска:** A0(-6,12), h = 0.09 epsilon = 0.00001, A=2, B = 10

**Таблица:**

**График траектории поиска:** 

**Результаты исследования:**

При значениях шага 0.09, радиусов эллипсоида А = 2 B = 10, координаты начальной точки A0(-6, 12) – метод случайного поиска достиг минимального значения функции в 0,00001195 в точках x = 0,0057 y = 0,0197 за 671 итераций. Видно, что при выборе других радиусов (значение второго радиуса сильно отличается от первого) и при выборе начальной точки, которая далеко от минимума метод потребовал больше итераций и показал медленную сходимость.

**Итоговая сводная таблица тестов функции эллипсоида:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **input** | | | | | | **output** | | | |
| **№ Теста** | **A0** | **h** | **e** | **A** | **B** | **x** | **y** | **F** | **Итерация** |
| №3 | {5; -8} | 0.06 | 0.001 | 4 | 3 | 0.8302 | 0.1613 | 0.04597 | 498 |
| №4 | {12;-1} | 0.001 | 0.0001 | 1 | 13 | 10,7850 | -0,9973 | 116,321 | 3793 |
| №5 | {-2.25, 6.95} | 0.07 | 0.001 | 5 | 6 | -2,0122 | 6,2947 | 1,26262 | 23 |
| №6 | {-1.25, 3.95} | 0.07 | 0.001 | 1 | 1 | -0,0080 | 0,0208 | 0,00049827 | 168 |

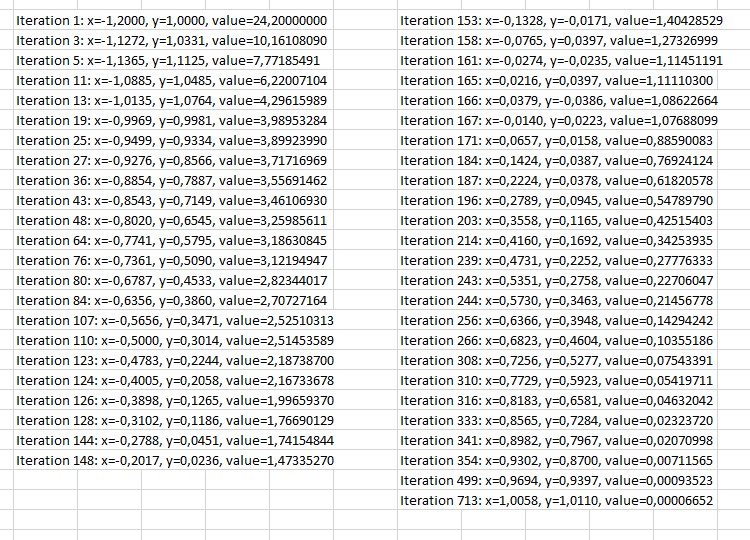
Выводы: после проведения нескольких тестов на функции эллипсоида, видно, что метод случайного поиска может успешно находить минимум функции, однако он не гарантирует нахождение глобального минимума. Для того, чтобы эффективно применять данный мтод, нужно правильно выбирать параметры (начальную точку, шаг, точность) и учитывать особенности тестируемой функции на наличие локальных минимумов.

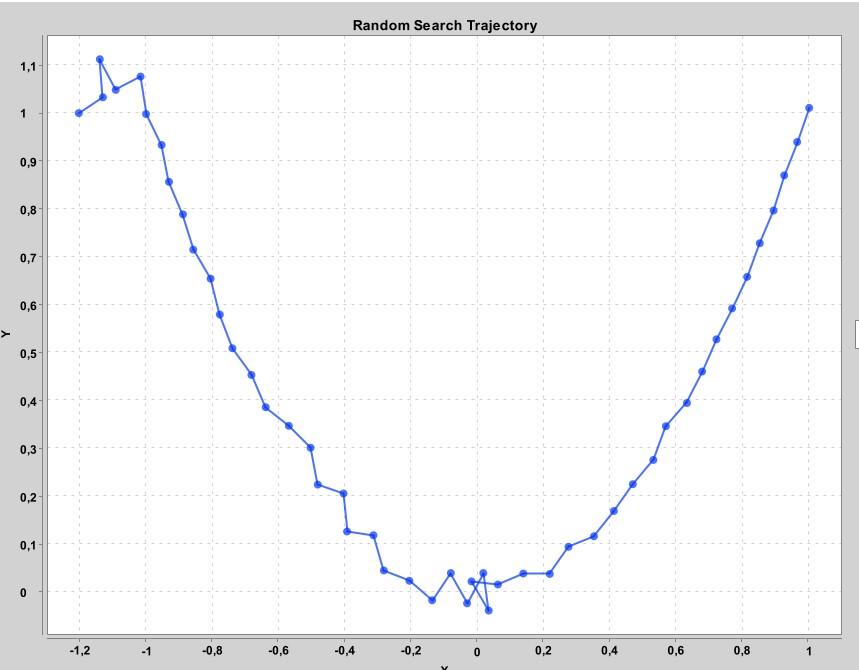
1. **Тестировании на функции Розенброка:**

**Функция Розенброка:**

**Параметры поиска:** A0(-1.2, 1), h = 0.06, epsilon = 0.00001

Таблица:



**График траектории поиска:** 

**Результаты исследования:**

При значениях шага 0.06, координаты начальной точки A0(-1.2, 1) – метод случайного поиска достиг минимального значения функции в 0,00006652 в точках x = 1,0058 y = 1,0110 за 713 итераций.

**Итоговая сводная таблица тестов функции Розенброка:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **input** | | | | **output** | | | |
| **№** | **A0** | **h** | **e** | **x** | **y** | **f** | **Итерация** |
| №2 | {4, 1} | 0.09 | 0.001 | 0,9838 | 0,9662 | 0,00051913 | 469 |
| №3 | {-1, 10} | 0.05 | 0.001 | 0,9972 | 0,9923 | 0,00047911 | 5136 |
| №4 | {-0.2, 0} | 0.06 | 0.0001 | 1,0007 | 1,0007 | 0,00004629 | 700 |

**Вывод:** согласно проведенным тестам функции Розенброка видно, что метод случайного поиска находит минимум функции, но скорость сходимости довольно низкая, потому что функция Розенброка имеет сложную форму. Точность и скорость нахождения минимума зависит от выбранных параметров: начальной точки, шага итд.

**Код:**

import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
import org.knowm.xchart.\*;  
import org.knowm.xchart.style.lines.SeriesLines;  
import org.knowm.xchart.style.markers.SeriesMarkers;  
  
public class RandomSearch {  
 private static double *A* = 1;  
 private static double *B* = 1;  
  
 public static double ellipsoid(double x, double y) {  
 return Math.*pow*(x / *A*, 2) + Math.*pow*(y / *B*, 2);  
 }  
  
public static double rosenbrock(double x, double y) {  
 return (1 - x) \* (1 - x) + 100 \* (y - x \* x) \* (y - x \* x);  
 }  
  
public static List<double[]> randomSearch(double lowerBoundX, double upperBoundX, double lowerBoundY, double upperBoundY, int iterations, boolean isEllipsoid, double[] startPoint, double stepSize, double epsilon) {  
 List<double[]> searchPoints = new ArrayList<>();  
 Random random = new Random();  
  
 double bestX = startPoint[0];  
 double bestY = startPoint[1];  
 double bestValue = isEllipsoid ? *ellipsoid*(bestX, bestY) : *rosenbrock*(bestX, bestY);  
  
 searchPoints.add(new double[]{bestX, bestY});  
 System.*out*.printf("Iteration 1: x=%.4f, y=%.4f, value=%.8f%n", bestX, bestY, bestValue);  
  
 for (int i = 1; i < iterations; i++) {  
 //double x = bestX + stepSize \* (random.nextDouble() \* 2 - 1);  
 //double y = bestY + stepSize \* (random.nextDouble() \* 2 - 1);  
 double angle = random.nextDouble() \* 2 \* Math.*PI*;  
 double x = bestX + stepSize \* Math.*cos*(angle);  
 double y = bestY + stepSize \* Math.*sin*(angle);  
 double currentValue = isEllipsoid ? *ellipsoid*(x, y) : *rosenbrock*(x, y);  
  
 if (currentValue < bestValue) {  
 double diff = bestValue - currentValue;  
 if (diff < epsilon) {  
 break;  
 }  
 bestX = x;  
 bestY = y;  
 bestValue = currentValue;  
 searchPoints.add(new double[]{bestX, bestY});  
 System.*out*.printf("Iteration %d: x=%.4f, y=%.4f, value=%.8f%n", i + 1, bestX, bestY, bestValue);  
 }  
 else if (i == iterations - 1 && searchPoints.size() == 1) {  
 System.*out*.println("Search ended: no better point ");  
 break;  
 }  
 else if (i == iterations - 1) {  
 System.*out*.println("Search ended: limit reached.");  
 break;  
 }  
 }  
 return searchPoints;  
 }  
  
public static void main(String[] args) {  
  
 double lowerBoundX = -1000;  
 double upperBoundX = 1000;  
 double lowerBoundY = -1000;  
 double upperBoundY = 1000;  
 int iterations = 10000;  
  
 //true для эллипсоида (0,0) , false для Розенброка (1.1)  
 boolean isEllipsoidFunction = false;  
  
 double[] startPoint = {-0.2, 0};  
 double stepSize = 0.06;  
 double epsilon = 0.0001;  
  
 List<double[]> searchPoints = *randomSearch* (lowerBoundX, upperBoundX, lowerBoundY, upperBoundY, iterations, isEllipsoidFunction, startPoint, stepSize, epsilon);  
 double[] xData = new double[searchPoints.size()];  
 double[] yData = new double[searchPoints.size()];  
  
 for (int i = 0; i < searchPoints.size(); i++) {  
 xData[i] = searchPoints.get(i)[0];  
 yData[i] = searchPoints.get(i)[1];  
 }  
  
 // Создание и отображение графика  
 XYChart chart = new XYChartBuilder().width(800).height(600).title("Random Search Trajectory").xAxisTitle("X").yAxisTitle("Y").build();  
 XYSeries series = chart.addSeries("Search Points", xData, yData);  
 series.setMarker(SeriesMarkers.*CIRCLE*);  
 series.setLineStyle(SeriesLines.*SOLID*);  
  
 new SwingWrapper<>(chart).displayChart();  
 }  
 }