**Министерство науки и высшего образования Российской Федерации**

федеральное государственное автономное образовательное учреждение высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Отчет**

по лабораторной работе №1 «Вариант 2. Океаническое дно»

по дисциплине «**Прикладная математика**»

Автор: Константинова Ольга Алексеевна

Факультет: Информационные технологии и программирование

Группа: M32111



Санкт-Петербург, 2023

**Цель работы:** изучение методов одномерного поиска нулевого порядка – метода дихотомии, метода золотого сечения, метода Фибоначчи, метода парабол и комбинированного метода Брента.

**Задачи:**

1. Решить задачу в соответствии с номером варианта (№2): реализовать алгоритмы одномерной минимизации функции без производной.
2. Сравнить методы по количеству итераций и количеству вычислений функции в зависимости от разной точности.
3. Протестировать реализованные алгоритмы для задач минимизации многомодальной функции.

Условие варианта №2 «Океаническое дно»

Исследовательская экспедиция на батискафе решила исследовать флору и фауну океанического желоба вблизи западного побережья Южной Америки. Ранее при помощи акустического профилирования была получена картина океанического дна на пути следования батискафа. Профиль дна описывается следующей функцией на заданном промежутке (в относительных единицах):

= 3 \*

Для построения точного курса исследователям необходимо извлечь из этих данных координату самой глубокой точки. Именно она будет являться океаническим желобом, который они так жаждут исследовать.

**Общие вводные данные**

Функция, описывающая профиль морского дна: = 3 \*

Интервал, на котором изучается функция: [π, 2π] (т. к. на нём функция является унимодальной).

Изображение выглядит как диаграмма

Автоматически созданное описание



(ниже на графике находится искомый минимум изучаемого отрезка)

Реализация методов на языке программирования java:

import java.util.ArrayList;  
import java.util.List;  
  
public class Main {  
 public static void main(String[] args)  
 {  
 // начальные значения интервала поиска, в котором функция является унимодальной  
 double a = Math.PI;  
 double b = Math.PI \* 2;  
  
 // заданная точность поиска  
 double e = 0.001;  
  
 dichotomy(a, b, e);  
 System.out.println();  
 goldenCut(a, b, e);  
 System.out.println();  
 fibonacci(a, b, e);  
 System.out.println();  
 quadraticInterpolation(a, b, e);  
 System.out.println();  
 brent(a, b, e);  
 }

Вспомогательные функции для подсчёта значения функции и вывода их в консоль:

private static double function(double x) // метод, который вычисляет значение функции в точке x  
{  
 return x \* x \* x \* Math.*sin*(x);  
}  
  
private static void printInterval(int n, double a, double b)  
{  
 System.*out*.println("Итерация №" + n + ":\t" + a + "\t" + b);  
}

Метод дихотомии

Точки x1, x2 выбираются на расстоянии δ < ε/2 от середины отрезка, где ε – заданная точность. На каждой итерации функция вычисляется 2 раза.

private static void dichotomy(double a, double b, double e)  
{  
 System.*out*.println("Метод дихотомии на отрезке [" + a + ", " + b + "]");  
 System.*out*.println("Таблица изменения интервала поиска:");  
  
 int n = 0; // количество итераций  
 int m = 0; // количество вычислений функции  
  
 final double delta = e / 3; // удовлетворяет условию < (e / 2)  
  
 while (b - a > e)  
 {  
 n++;  
 *printInterval*(n, a, b);  
  
 double mid = (a + b) / 2; // середина отрезка  
  
 double x1 = mid - delta;  
 double x2 = mid + delta;  
 double fx1 = *function*(x1);  
 double fx2 = *function*(x2);  
 m += 2;  
  
 if (fx1 > fx2)  
 {  
 a = x1;  
 }  
 else if (fx1 < fx2)  
 {  
 b = x2;  
 }  
 else  
 {  
 a = x1;  
 b = x2;  
 }  
 }  
  
 System.*out*.println("Минимум находится в интервале: [" + a + ", " + b + "]");  
 System.*out*.println("Значение функции = " + *function*((a + b) / 2));  
 System.*out*.println("Количество итераций: " + n);  
 System.*out*.println("Количество вычислений функции: " + m);  
}

При ε = 0,1 количество итераций составляет 7:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.6790556470513565 | 6.283185307179586 |
| Итерация №3: | 4.6790556470513565 | 5.514453810448805 |
| Итерация №4: | 5.063421395416747 | 5.514453810448805 |
| Итерация №5: | 5.063421395416747 | 5.32227093626611 |
| Итерация №6: | 5.159512832508096 | 5.32227093626611 |
| Итерация №7: | 5.159512832508096 | 5.274225217720436 |
| Минимум находится в интервале: | 5.183535691780932 | 5.274225217720436 |

Количество вычислений функции: 14

Найденное значение функции = -124.31521055144607

При ε = 0,01 количество итераций составляет 10:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.709055647051357 | 6.283185307179586 |
| Итерация №3: | 4.709055647051357 | 5.499453810448805 |
| Итерация №4: | 5.100921395416748 | 5.499453810448805 |
| Итерация №5: | 5.100921395416748 | 5.30352093626611 |
| Итерация №6: | 5.198887832508096 | 5.30352093626611 |
| Итерация №7: | 5.198887832508096 | 5.254537717720436 |
| Итерация №8: | 5.223379441780933 | 5.254537717720436 |
| Итерация №9: | 5.223379441780933 | 5.242291913084018 |
| Итерация №10: | 5.229502344099143 | 5.242291913084018 |
| Минимум находится в интервале: | 5.229502344099143 | 5.2392304619249135 |

Количество вычислений функции: 20

Найденное значение функции = -124.3164982608352

При ε = 0,001 количество итераций составляет 14

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.712055647051356 | 6.283185307179586 |
| Итерация №3: | 4.712055647051356 | 5.497953810448805 |
| Итерация №4: | 5.104671395416746 | 5.497953810448805 |
| Итерация №5: | 5.104671395416746 | 5.301645936266109 |
| Итерация №6: | 5.202825332508094 | 5.301645936266109 |
| Итерация №7: | 5.202825332508094 | 5.2525689677204355 |
| Итерация №8: | 5.227363816780931 | 5.2525689677204355 |
| Итерация №9: | 5.227363816780931 | 5.240299725584017 |
| Итерация №10: | 5.227363816780931 | 5.234165104515808 |
| Итерация №11: | 5.230431127315036 | 5.234165104515808 |
| Итерация №12: | 5.2319647825820885 | 5.234165104515808 |
| Итерация №13: | 5.2319647825820885 | 5.233398276882282 |
| Итерация №14: | 5.232348196398851 | 5.233398276882282 |
| Минимум находится в интервале: | 5.232539903307233 | 5.233398276882282 |

Количество вычислений функции: 28

Найденное значение функции = -124.316680550793

Метод золотого сечения

На первой итерации точки x1, x2 выбираются по принципу золотого сечения:

1

2 , где

.

Функция вычисляется дважды.

На последующих итерациях одна из точек x1, x2 с предыдущей итерации используется повторно, как и вычисленное для неё значение функции. Скорость уменьшения интервала неопределенности ниже, чем в методе дихотомии, но значение функции вычислять нужно лишь один раз на итерацию.

private static void goldenCut(double a, double b, double e)  
{  
 System.*out*.println("Метод золотого сечения на отрезке [" + a + ", " + b + "]");  
 System.*out*.println("Таблица изменения интервала поиска:");  
  
 int n = 0; // количество итераций  
 int m = 0; // количество вычислений функции  
  
 final double goldenNumber = (Math.*sqrt*(5) - 1) / 2; // золотое число  
  
 boolean left = false; // флаг, обозначающий, что нужно вычислить местонахождение левой точки и значение ф-и в ней  
 double fxPrev = 0; // значение ф-и, вычисленное на предыдущей итерации, для x, к-е остаётся для текущей итерации  
  
 double x1 = b - (b - a) \* goldenNumber;  
 double x2 = a + (b - a) \* goldenNumber;  
 n = 1; // произведена первая итерация  
  
 double fx1 = *function*(x1);  
 double fx2 = *function*(x2);  
 m = 2; // произведён подсчёт значения функции дважды  
  
 *printInterval*(n, a, b);  
  
 if (fx1 > fx2)  
 {  
 a = x1;  
 left = false;  
 fxPrev = fx2;  
 }  
 else if (fx1 <= fx2)  
 {  
 b = x2;  
 left = true;  
 fxPrev = fx1;  
 }  
  
 while (b - a > e)  
 {  
 n++; // очередная итерация  
 m++; // очередной подсчёт происходит единожды  
 *printInterval*(n, a, b);  
 if (left)  
 {  
 x2 = x1; // точка, оставшаяся с предыдущей итерации, становится правой точкой  
 fx2 = fxPrev;  
  
 x1 = b - (b - a) \* goldenNumber; // вычисляем местоположение левой точки  
 fx1 = *function*(x1);  
 }  
 else  
 {  
 x1 = x2; // точка, оставшаяся с предыдущей итерации, становится левой точкой  
 fx1 = fxPrev;  
  
 x2 = a + (b - a) \* goldenNumber; // вычисляем местоположение правой точки  
 fx2 = *function*(x2);  
 }  
  
 if (fx1 > fx2)  
 {  
 a = x1;  
 left = false;  
 fxPrev = fx2;  
 }  
 else if (fx1 <= fx2)  
 {  
 b = x2;  
 left = true;  
 fxPrev = fx1;  
 }  
 }  
  
 System.*out*.println("Минимум находится в интервале: [" + a + ", " + b + "]");  
 System.*out*.println("Значение функции = " + *function*((a + b) / 2));  
 System.*out*.println("Количество итераций: " + n);  
 System.*out*.println("Количество вычислений функции: " + m);  
}

При ε = 0,1 количество итераций составляет 8:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.34157426845412 | 6.283185307179586 |
| Итерация №3: | 4.34157426845412 | 5.541555883318447 |
| Итерация №4: | 4.799926459457307 | 5.541555883318447 |
| Итерация №5: | 5.0832036923152595 | 5.541555883318447 |
| Итерация №6: | 5.0832036923152595 | 5.366480925173213 |
| Итерация №7: | 5.191405967027979 | 5.366480925173213 |
| Итерация №8: | 5.191405967027979 | 5.299608241740699 |
| Минимум находится в интервале: | 5.191405967027979 | 5.258278650460493 |

Количество вычислений функции: 9

Найденное значение функции = -124.31083717472528

При ε = 0,01 количество итераций составляет 12:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.34157426845412 | 6.283185307179586 |
| Итерация №3: | 4.34157426845412 | 5.541555883318447 |
| Итерация №4: | 4.799926459457307 | 5.541555883318447 |
| Итерация №5: | 5.0832036923152595 | 5.541555883318447 |
| Итерация №6: | 5.0832036923152595 | 5.366480925173213 |
| Итерация №7: | 5.191405967027979 | 5.366480925173213 |
| Итерация №8: | 5.191405967027979 | 5.299608241740699 |
| Итерация №9: | 5.191405967027979 | 5.258278650460493 |
| Итерация №10: | 5.216949059180288 | 5.258278650460493 |
| Итерация №11: | 5.216949059180288 | 5.242492151332596 |
| Итерация №12: | 5.226705652204699 | 5.242492151332596 |
| Минимум находится в интервале: | 5.226705652204699 | 5.236462245229109 |

Количество вычислений функции: 13

Найденное значение функции = -124.31651669540263

При ε = 0,001 количество итераций составляет 17:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.34157426845412 | 6.283185307179586 |
| Итерация №3: | 4.34157426845412 | 5.541555883318447 |
| Итерация №4: | 4.799926459457307 | 5.541555883318447 |
| Итерация №5: | 5.0832036923152595 | 5.541555883318447 |
| Итерация №6: | 5.0832036923152595 | 5.366480925173213 |
| Итерация №7: | 5.191405967027979 | 5.366480925173213 |
| Итерация №8: | 5.191405967027979 | 5.299608241740699 |
| Итерация №9: | 5.191405967027979 | 5.258278650460493 |
| Итерация №10: | 5.216949059180288 | 5.258278650460493 |
| Итерация №11: | 5.216949059180288 | 5.242492151332596 |
| Итерация №12: | 5.226705652204699 | 5.242492151332596 |
| Итерация №13: | 5.226705652204699 | 5.236462245229109 |
| Итерация №14: | 5.230432339125624 | 5.236462245229109 |
| Итерация №15: | 5.230432339125624 | 5.234159026046548 |
| Итерация №16: | 5.2318558068639875 | 5.234159026046548 |
| Итерация №17: | 5.2318558068639875 | 5.233279274602351 |
| Минимум находится в интервале: | 5.232399523158153 | 5.233279274602351 |

Количество вычислений функции: 18

Найденное значение функции = -124.31667975758243

Метод Фибоначчи

Находим необходимое количество чисел Фибоначчи – n штук, так чтобы длина исходного отрезка, деленная на самое большое из них, была < заданной точности ε.

Метод предполагает выполнение n – 1 шага. По окончание алгоритма последний интервал неопределенности окажется < ε. Функция будет вычислена всего n раз.

На первом шаге вычисляем x1, x2 следующим образом:

1 = n-2 n

2 n-1 n

Функция вычисляется дважды.

На последующих итерациях одна из точек x1, x2 с предыдущей итерации используется повторно, как и вычисленное для неё значение функции. Скорость уменьшения интервала неопределенности разная для каждого шага алгоритма. На k-ой итерации интервал неопределенности сжимается с коэффициентом **Fn – k / Fn – k + 1.**

Значение функции вычислять нужно лишь один раз на итерацию, как и в случае метода золотого сечения (метод Фибоначчи является улучшением метода золотого сечения).

private static void fibonacci(double a, double b, double e)  
{  
 System.*out*.println("Метод фибоначчи на отрезке [" + a + ", " + b + "]");  
 System.*out*.println("Таблица изменения интервала поиска:");  
  
 int n = 0; // количество итераций  
  
 List<Double> fibonacciNumbers = new ArrayList<>();  
 fibonacciNumbers.add(1.0);  
 fibonacciNumbers.add(1.0);  
 n = 2;  
  
 // найдём n - необходимое количество итераций, нужное для метода Фибоначчи (за n шагов интервал неопределенности  
 // должен уменьшится до e)  
 while (true)  
 {  
 double fn = fibonacciNumbers.get(n - 2) + fibonacciNumbers.get(n - 1);  
 fibonacciNumbers.add(fn);  
 n++;  
 if ((b - a) / fn < e)  
 {  
 // найдено достаточно большое число Фибоначчи  
 break;  
 }  
 }  
  
 // нулевой шаг (k = 0)  
 *printInterval*(1, a, b);  
 double x1 = a + fibonacciNumbers.get((n - 1) - 2) / fibonacciNumbers.get(n - 1) \* (b - a);  
 double x2 = a + fibonacciNumbers.get((n - 1) - 1) / fibonacciNumbers.get(n - 1) \* (b - a);  
  
 double fx1 = *function*(x1);  
 double fx2 = *function*(x2);  
  
 boolean left = false; // флаг, обозначающий, что нужно вычислить местонахождение левой точки и значение ф-и в ней  
 double fxPrev = 0; // значение ф-и, вычисленное на предыдущей итерации, для x, к-е остаётся для текущей итерации  
  
 if (fx1 > fx2)  
 {  
 a = x1;  
 left = false;  
 fxPrev = fx2;  
 }  
 else if (fx1 <= fx2)  
 {  
 b = x2;  
 left = true;  
 fxPrev = fx1;  
 }  
  
 for (int k = 1; k < n - 2; k++)  
 {  
 *printInterval*(k + 1, a, b);  
 if (left)  
 {  
 x2 = x1; // точка, оставшаяся с предыдущей итерации, становится правой точкой  
 fx2 = fxPrev;  
  
 x1 = a + fibonacciNumbers.get((n - 1) - k - 2) / fibonacciNumbers.get((n - 1) - k) \* (b - a); // вычисляем местоположение левой точки  
 fx1 = *function*(x1);  
 }  
 else  
 {  
 x1 = x2; // точка, оставшаяся с предыдущей итерации, становится левой точкой  
 fx1 = fxPrev;  
  
 x2 = a + fibonacciNumbers.get((n - 1) - k - 1) / fibonacciNumbers.get((n - 1) - k) \* (b - a); // вычисляем местоположение правой точки  
 fx2 = *function*(x2);  
 }  
  
 if (fx1 > fx2)  
 {  
 a = x1;  
 left = false;  
 fxPrev = fx2;  
 }  
 else if (fx1 <= fx2)  
 {  
 b = x2;  
 left = true;  
 fxPrev = fx1;  
 }  
 }  
  
 *printInterval*(n - 1, a, b);  
  
 System.*out*.println("Минимум находится в интервале: [" + a + ", " + b + "]");  
 System.*out*.println("Значение функции = " + *function*((a + b) / 2));  
 System.*out*.println("Количество итераций: " + (n - 1));  
 System.*out*.println("Количество вычислений функции: " + n);  
}

При ε = 0,1 количество итераций составляет 8:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.342789844668244 | 6.283185307179586 |
| Итерация №3: | 4.342789844668244 | 5.543987035746694 |
| Итерация №4: | 4.804788764313802 | 5.543987035746694 |
| Итерация №5: | 5.081988116101136 | 5.543987035746694 |
| Итерация №6: | 5.081988116101136 | 5.359187467888471 |
| Итерация №7: | 5.174387900030247 | 5.359187467888471 |
| Итерация №8: | 5.174387900030247 | 5.266787683959359 |
| Минимум находится в интервале: | 5.174387900030247 | 5.266787683959359 |

Количество вычислений функции: 9

Найденное значение функции = -124.30310202462606

При ε = 0,01 количество итераций составляет 13:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.341564383342924 | 6.283185307179586 |
| Итерация №3: | 4.341564383342924 | 5.541536113096054 |
| Итерация №4: | 4.799886919012522 | 5.541536113096054 |
| Итерация №5: | 5.083213577426456 | 5.541536113096054 |
| Итерация №6: | 5.083213577426456 | 5.3665402358403895 |
| Итерация №7: | 5.191544358584725 | 5.3665402358403895 |
| Итерация №8: | 5.191544358584725 | 5.299875139742993 |
| Итерация №9: | 5.191544358584725 | 5.25820945468212 |
| Итерация №10: | 5.216543769621248 | 5.25820945468212 |
| Итерация №11: | 5.216543769621248 | 5.2415431806577715 |
| Итерация №12: | 5.224876906633423 | 5.2415431806577715 |
| Итерация №13: | 5.224876906633423 | 5.233210043645597 |
| Минимум находится в интервале: | 5.224876906633423 | 5.233210043645597 |

Количество вычислений функции: 14

Найденное значение функции = -124.31532621711257

При ε = 0,001 количество итераций составляет 18:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.341574348826076 | 6.283185307179586 |
| Итерация №3: | 4.341574348826076 | 5.541556044062359 |
| Итерация №4: | 4.799926780945132 | 5.541556044062359 |
| Итерация №5: | 5.083203611943303 | 5.541556044062359 |
| Итерация №6: | 5.083203611943303 | 5.366480442941474 |
| Итерация №7: | 5.191404841820589 | 5.366480442941474 |
| Итерация №8: | 5.191404841820589 | 5.299606071697874 |
| Итерация №9: | 5.191404841820589 | 5.258279213064188 |
| Итерация №10: | 5.216952354430504 | 5.258279213064188 |
| Итерация №11: | 5.216952354430504 | 5.242499867040418 |
| Итерация №12: | 5.226720521016647 | 5.242499867040418 |
| Итерация №13: | 5.226720521016647 | 5.236488687602791 |
| Итерация №14: | 5.230477508165164 | 5.236488687602791 |
| Итерация №15: | 5.230477508165164 | 5.234234495313681 |
| Итерация №16: | 5.231980303024571 | 5.234234495313681 |
| Итерация №17: | 5.231980303024571 | 5.233483097883977 |
| Итерация №18: | 5.2327317004542735 | 5.233483097883977 |
| Минимум находится в интервале: | 5.2327317004542735 | 5.233483097883977 |

Количество вычислений функции: 19

Найденное значение функции = -124.31667808292161

Метод парабол

(метод квадратичной интерполяции)

Задаём начальную точку x, находящуюся в исходном интервале – эта точка и края интервала образуют точки на графике функции, которые мы вычисляем. Через полученные три точки проводим параболу и находим точку её минимума u.

На основе четырёх точек, сравнивая значения функции в них, выбираем новый интервал неопределенности, который задаётся двумя из четырех точек, и третья точка между ними становится новой точкой x. Процесс повторяется до тех пор, пока интервал неопределенности не станет меньше заданной точности ε.

Метод парабол обладает высокой скоростью сходимости, однако при некоторых условиях сходимость не выполняется.

private static void quadraticInterpolation(double a, double b, double e)  
{  
 System.*out*.println("Метод парабол на отрезке [" + a + ", " + b + "]");  
 System.*out*.println("Таблица изменения интервала поиска:");  
  
 int n = 0; // количество итераций  
 int m = 0; // количество вычислений функции  
  
 double x = a + (b - a) / 3;  
  
 double fa = *function*(a);  
 double fx = *function*(x);  
 double fb = *function*(b);  
 m = 3;  
  
 while (b - a > e)  
 {  
 n++;  
 *printInterval*(n, a, b);  
  
 // вычисляем минимум параболы, проходящей через точки a, b, x на графике  
 double u = x - ((x - a) \* (x - a) \* (fx - fb) - (x - b) \* (x - b) \* (fx - fa)) / (2 \* ((x - a) \* (fx - fb) - (x - b) \* (fx - fa)));  
 double fu = *function*(u);  
 m++;  
  
 // находим следующий интервал неопределенности  
 if (a <= u && u <= x)  
 {  
 if (fu >= fx) // минимум должен находится в промежутке [u, b]  
 {  
 a = u;  
 fa = fu;  
 }  
 else // минимум должен находится в промежутке [a, x]  
 {  
 b = x;  
 fb = fx;  
 x = u;  
 fx = fu;  
 }  
 }  
 else if (x <= u && u <= b)  
 {  
 if (fu >= fx) // минимум должен находится в промежутке [a, u]  
 {  
 b = u;  
 fb = fu;  
 }  
 else // минимум должен находится в промежутке [x, b]  
 {  
 a = x;  
 fa = fx;  
 x = u;  
 fx = fu;  
 }  
 }  
 else  
 {  
 break;  
 }  
 }  
  
 System.*out*.println("Минимум находится в интервале: [" + a + ", " + b + "]");  
 System.*out*.println("Значение функции = " + *function*((a + b) / 2));  
 System.*out*.println("Количество итераций: " + n);  
 System.*out*.println("Количество вычислений функции: " + m);  
}

При ε = 0,1 количество итераций составляет 21:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.1887902047863905 | 6.283185307179586 |
| Итерация №3: | 4.71238898038469 | 6.283185307179586 |
| Итерация №4: | 5.016380758583049 | 6.283185307179586 |
| Итерация №5: | 5.141963644783866 | 6.283185307179586 |
| Итерация №6: | 5.198448787261813 | 6.283185307179586 |
| Итерация №7: | 5.219485761549047 | 6.283185307179586 |
| Итерация №8: | 5.227911128794368 | 6.283185307179586 |
| Итерация №9: | 5.231016930137759 | 6.283185307179586 |
| Итерация №10: | 5.2322184213652525 | 6.283185307179586 |
| Итерация №11: | 5.232665087137704 | 6.283185307179586 |
| Итерация №12: | 5.232835700811969 | 6.283185307179586 |
| Итерация №13: | 5.232899556597859 | 6.283185307179586 |
| Итерация №14: | 5.23292380563888 | 6.283185307179586 |
| Итерация №15: | 5.2329329166349305 | 6.283185307179586 |
| Итерация №16: | 5.232936366310619 | 6.283185307179586 |
| Итерация №17: | 5.2329376651312245 | 6.283185307179586 |
| Итерация №18: | 5.232938156154778 | 6.283185307179586 |
| Итерация №19: | 5.232938341144728 | 6.283185307179586 |
| Итерация №20: | 5.2329384112098145 | 6.283185307179586 |
| Итерация №21: | 5.232938436463308 | 6.283185307179586 |
| Минимум находится в интервале: | 5.232938436463308 | 5.232938452380226 |

Количество вычислений функции: 24

Найденное значение функции = -124.31668063470212

При ε = 0,01 количество итераций составляет 21:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.1887902047863905 | 6.283185307179586 |
| Итерация №3: | 4.71238898038469 | 6.283185307179586 |
| Итерация №4: | 5.016380758583049 | 6.283185307179586 |
| Итерация №5: | 5.141963644783866 | 6.283185307179586 |
| Итерация №6: | 5.198448787261813 | 6.283185307179586 |
| Итерация №7: | 5.219485761549047 | 6.283185307179586 |
| Итерация №8: | 5.227911128794368 | 6.283185307179586 |
| Итерация №9: | 5.231016930137759 | 6.283185307179586 |
| Итерация №10: | 5.2322184213652525 | 6.283185307179586 |
| Итерация №11: | 5.232665087137704 | 6.283185307179586 |
| Итерация №12: | 5.232835700811969 | 6.283185307179586 |
| Итерация №13: | 5.232899556597859 | 6.283185307179586 |
| Итерация №14: | 5.23292380563888 | 6.283185307179586 |
| Итерация №15: | 5.2329329166349305 | 6.283185307179586 |
| Итерация №16: | 5.232936366310619 | 6.283185307179586 |
| Итерация №17: | 5.2329376651312245 | 6.283185307179586 |
| Итерация №18: | 5.232938156154778 | 6.283185307179586 |
| Итерация №19: | 5.232938341144728 | 6.283185307179586 |
| Итерация №20: | 5.2329384112098145 | 6.283185307179586 |
| Итерация №21: | 5.232938436463308 | 6.283185307179586 |
| Минимум находится в интервале: | 5.232938436463308 | 5.232938452380226 |

Количество вычислений функции: 24

Найденное значение функции = -124.31668063470212

При ε = 0,001 количество итераций составляет 21:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.1887902047863905 | 6.283185307179586 |
| Итерация №3: | 4.71238898038469 | 6.283185307179586 |
| Итерация №4: | 5.016380758583049 | 6.283185307179586 |
| Итерация №5: | 5.141963644783866 | 6.283185307179586 |
| Итерация №6: | 5.198448787261813 | 6.283185307179586 |
| Итерация №7: | 5.219485761549047 | 6.283185307179586 |
| Итерация №8: | 5.227911128794368 | 6.283185307179586 |
| Итерация №9: | 5.231016930137759 | 6.283185307179586 |
| Итерация №10: | 5.2322184213652525 | 6.283185307179586 |
| Итерация №11: | 5.232665087137704 | 6.283185307179586 |
| Итерация №12: | 5.232835700811969 | 6.283185307179586 |
| Итерация №13: | 5.232899556597859 | 6.283185307179586 |
| Итерация №14: | 5.23292380563888 | 6.283185307179586 |
| Итерация №15: | 5.2329329166349305 | 6.283185307179586 |
| Итерация №16: | 5.232936366310619 | 6.283185307179586 |
| Итерация №17: | 5.2329376651312245 | 6.283185307179586 |
| Итерация №18: | 5.232938156154778 | 6.283185307179586 |
| Итерация №19: | 5.232938341144728 | 6.283185307179586 |
| Итерация №20: | 5.2329384112098145 | 6.283185307179586 |
| Итерация №21: | 5.232938436463308 | 6.283185307179586 |
| Минимум находится в интервале: | 5.232938436463308 | 5.232938452380226 |

Количество вычислений функции: 24

Найденное значение функции = -124.31668063470212

(дополнительное изображение параболы через точки a, b, x на первой итерации метода при ε = 0,001)



Комбинированный метод Брента

Данный метод призван улучшить результаты метода парабол.

На каждой итерации рассматриваем три точки x, w, v, x – точка с наименьшим значением функции из трёх, w -  вторая снизу, v – значение w с предыдущей итерации.

Если три точки различны, то на данной итерации применяется метод парабол: строим параболу через три точки, находим её минимум в точке u, проверяем что u не находится близко к краям интервала неопределенности, проверяем что u не оказалось слишком близко к точке x и отвергаем u, в случае если это не так. Если точки не были различными или найденное u не подошло, применяем метод золотого сечения для поиска нового значения u.

Для найденной любым способом точки u вычисляем значение функции и выбираем новые точки x, w, v, а также определяем новые границы интервала неопределенности.

На каждой итерации значение функции вычисляется только один раз – в выбранной точке u, остальные значения функции используются с предыдущей итерации.

private static void brent(double a, double c, double e)  
{  
 System.*out*.println("Комбинированный метод Брента на отрезке [" + a + ", " + c + "]");  
 System.*out*.println("Таблица изменения интервала поиска:");  
  
 int n = 0; // количество итераций  
 int m = 0; // количество вычислений функции  
  
 final double goldenNumber = (3 - Math.*sqrt*(5)) / 2; // золотое число  
  
 double x = (a + c) / 2;  
 double w = x;  
 double v = x;  
  
 double fx = *function*(x);  
 double fw = fx;  
 double fv = fx;  
 m++;  
  
 double d = c - a; // длина интервала текущего шага  
 double p = d; // длина интервала предыдущего шага  
  
 while (c - a > e)  
 {  
 n++;  
 *printInterval*(n, a, c);  
  
 boolean good = false; // флаг, обозначающий, что не нужно использовать золотое сечение  
 double u = Double.*MIN\_VALUE*;  
 double fu;  
  
 if (!(x == w || w == v || x == v || fx == fw || fw == fv || fx == fv))  
 {  
 // переходим к методу парабол  
  
 // вычисляем минимум параболы, проходящей через точки x, w, v на графике  
 u = x - ((x - w) \* (x - w) \* (fx - fv) - (x - v) \* (x - v) \* (fx - fw)) / (2 \* ((x - w) \* (fx - fv) - (x - v) \* (fx - fw)));  
  
 // проверим, подходит ли найденная точка u  
 if (a + e < u && c - e > u && Math.*abs*(u - x) < p / 2)  
 {  
 d = Math.*abs*(u - x);  
 good = true; // метод парабол сработал, золотое сечение не понадобится  
 }  
 }  
  
 if (!good) // используем ли метод золотого сечения?  
 {  
 if (x < (a + c) / 2) // левее середины интервала?  
 {  
 u = x + goldenNumber \* (c - x); // золотое сечение для отрезка [x, c]  
 d = c - x;  
 }  
 else  
 {  
 u = x - goldenNumber \* (x - a); // золотое сечение для отрезка [a, x]  
 d = x - a;  
 }  
 }  
  
 if (good && Math.*abs*(u - x) < e) // если u слишком близко к x, то отодвигаем её на точность e  
 {  
 // если u правее x, то сдинется вправо, иначе - влево  
 u = x + Math.*signum*(u - x) \* e;  
 }  
  
 fu = *function*(u);  
 m++;  
  
 // определим интервал неопределенности для следующей итерации  
 if (fu <= fx)  
 {  
 if (u >= x)  
 {  
 a = x;  
 }  
 else  
 {  
 c = x;  
 }  
  
 v = w;  
 w = x;  
 x = u;  
  
 fv = fw;  
 fw = fx;  
 fx = fu;  
 }  
 else  
 {  
 if (u >= x)  
 {  
 c = u;  
 }  
 else  
 {  
 a = u;  
 }  
  
 if (fu <= fw || w == x)  
 {  
 v = w;  
 w = u;  
 fv = fw;  
 fw = fu;  
 }  
 else if (fu <= fv || v == x || v == w)  
 {  
 v = u;  
 fv = fu;  
 }  
 }  
 }  
  
 System.*out*.println("Минимум находится в интервале: [" + a + ", " + c + "]");  
 System.*out*.println("Значение функции = " + *function*((a + c) / 2));  
 System.*out*.println("Количество итераций: " + n);  
 System.*out*.println("Количество вычислений функции: " + m);  
  
}

При ε = 0,1 количество итераций составляет 8:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.112398172952527 | 6.283185307179586 |
| Итерация №3: | 4.71238898038469 | 6.283185307179586 |
| Итерация №4: | 4.71238898038469 | 5.419233824128588 |
| Итерация №5: | 4.71238898038469 | 5.312379787816853 |
| Итерация №6: | 5.01923694426805 | 5.312379787816853 |
| Итерация №7: | 5.136442437092794 | 5.312379787816853 |
| Итерация №8: | 5.208879415326667 | 5.312379787816853 |
| Минимум находится в интервале: | 5.208879415326667 | 5.272846163373876 |

Количество вычислений функции: 9

Найденное значение функции = -124.31105167031359

При ε = 0,01 количество итераций составляет 10:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.112398172952527 | 6.283185307179586 |
| Итерация №3: | 4.71238898038469 | 6.283185307179586 |
| Итерация №4: | 4.71238898038469 | 5.419233824128588 |
| Итерация №5: | 4.71238898038469 | 5.312379787816853 |
| Итерация №6: | 5.208879415326667 | 5.312379787816853 |
| Итерация №7: | 5.224208312104474 | 5.312379787816853 |
| Итерация №8: | 5.224208312104474 | 5.264067158875865 |
| Итерация №9: | 5.224208312104474 | 5.24561337670627 |
| Итерация №10: | 5.224208312104474 | 5.238564659138472 |
| Минимум находится в интервале: | 5.230388651991973 | 5.238564659138472 |

Количество вычислений функции: 11

Найденное значение функции = -124.31646900330041

При ε = 0,001 количество итераций составляет 13:

|  |  |  |
| --- | --- | --- |
| Итерация №1: | 3.141592653589793 | 6.283185307179586 |
| Итерация №2: | 4.112398172952527 | 6.283185307179586 |
| Итерация №3: | 4.71238898038469 | 6.283185307179586 |
| Итерация №4: | 4.71238898038469 | 5.419233824128588 |
| Итерация №5: | 4.71238898038469 | 5.312379787816853 |
| Итерация №6: | 5.208879415326667 | 5.312379787816853 |
| Итерация №7: | 5.208879415326667 | 5.234208312104474 |
| Итерация №8: | 5.208879415326667 | 5.233631110535671 |
| Итерация №9: | 5.223558770256259 | 5.233631110535671 |
| Итерация №10: | 5.22916578490644 | 5.233631110535671 |
| Итерация №11: | 5.231307473927391 | 5.233631110535671 |
| Итерация №12: | 5.232125526340061 | 5.233631110535671 |
| Итерация №13: | 5.232631110535671 | 5.233631110535671 |
| Минимум находится в интервале: | 5.232631110535671 | 5.233249144524422 |

Количество вычислений функции: 14

Найденное значение функции = -124.3166806344516

**Выводы**

В ходе выполнения лабораторной работы были изучены методы одномерного поиска нулевого порядка (без вычисления производных). Общее между всеми алгоритмами заключается в том, что мы стремимся уменьшить интервал неопределенности – отрезок, на котором мы в данный момент изучаем функцию, – до необходимого минимума – точности ε. Об эффективности алгоритмов можно судить по числу вычислений функции, необходимому для достижения заданной точности.

Наиболее эффективными оказались методы золотого сечения, Фибоначчи, Брента: например, при ε = 0,1 им понадобилось 9 вычислений функции. Метод парабол показал себя неэффективным – ему потребовалось 24 вычислений. Метод дихотомии также проиграл тройке победителей – ему всегда нужно 2 вычисления функции на каждой итерации, кроме первой.

Для более высокой точности ε = 0,001 метод Брента показал себя лучше всех.

Тестирование многомодальной функции

Была протестирована та же функция: = 3 \* на интервале на которой она не является унимодальной – имеет два экстремума со значениями: -124.316 и -1378.019.

Все реализованные методы на этом интервале смогли найти первый экстремум, который не является минимумом функции.