Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 1

по курсу «Методы оптимизации»

«Проверка функции на унимодальность и поиск её минимума различными способами»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

1. Реализовать метод проверки функции на унимодальность.

2. Реализовать поиск минимума унимодальной функции на полученном интервале методами прямого перебора, дихотомии, золотого сечения, фибоначчи.

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```
1
2  function f(x)
3      return x^2
4  end
5
6  function f_deriv(f, x; h=1e-5)
7      return (f(x + h) - f(x - h)) / (2h)
8  end
9
10 function f_deriv_2(f, x; h=1e-5)
11     return (f(x + h) - 2 * f(x) + f(x - h)) / (h^2)
12 end
13
14 using PyPlot
15
16 x_vals = -10:0.1:10
17 y_vals = f.(x_vals)
18
19 PyPlot.figure(figsize=(6, 5))
20 PyPlot.plot(x_vals, y_vals, label="f(x)")
21 PyPlot.grid()
22 PyPlot.legend()
23 PyPlot.show()
24
25 function is_unimodal_1(f, a, b, n)
26     step=(b-a)/n
27     x_values = a:step:b
28     derivative_values = [f_deriv(f, x) for x in x_values]
29
30     for i in 1:(length(derivative_values) - 1)
31         if derivative_values[i] > derivative_values[i + 1]
32             return false
```

```
        end
    end
    return true
end

println("                                                    1: ",
    is_unimodal_1(f,-10,-7,1000))

function is_unimodal_2(f, a, b, n)
    step=(b-a)/n
    x_values = a:step:b
    for x in x_values
        if f_deriv_2(f,x) < 0
            return false
        end
    end
    return true
end

print("                                                    2: ",
    is_unimodal_2(f,-10,-7,1000))

function find_l(f, a, b, n)
    step=(b-a)/n
    x_values = a:step:b
    derivative_values = [f_deriv(f, x) for x in x_values]

    maxVal = -100000
    for i in 1:(length(derivative_values) - 1)
        if derivative_values[i] > maxVal
            maxVal=derivative_values[i]
        end
    end

    return maxVal
end

L=find_l(f,-10,-7,1000)

function search(f, a, b, epsilon)
    h = epsilon
    n = Int(floor((b - a) / h))
    x_values = [a + i * h for i in 0:n]
    f_values = f.(x_values)

    f_min, idx = findmin(f_values)
```

```
    x_min = x_values[idx]

    return x_min, f_min, length(x_values)
end

using Plots

a = -10
b = -7
acc = 0.01

x_search, y_search, iters_search = search(f, a, b, acc)

println("x = ",x_search)
println("y = ",y_search)
println("y = ",iters_search)

xs = a:0.1:b
ys = f.(xs)

Plots.plot(xs, ys, label="f(x)", xlabel="x", ylabel="f(x)", color=:
    purple)

scatter!([x_search], [y_search], label="                                Min(
    eps=$acc)", color=:red)
Plots.plot!()

function bisection(f, a, b, eps)
    a = Float64(a)
    b = Float64(b)
    iters = 0
    delta = rand(0:2*eps)
    while b - a > eps
        iters += 1
        x1 = (a + b - delta) / 2
        x2 = (a + b + delta) / 2
        if f(x1) < f(x2)
            b = x2
        else
            a = x1
        end
    end

    min = (a + b) / 2
    pogr = L*(a-b)/2
    return min, f(min), iters, pogr
```

```
121 end
122
123 using Plots
124
125 a = -10
126 b = -7
127 acc = 0.01
128
129 x_bisection, y_bisection, iters_bisection, pogr_bisection = bisection(f,
        a, b, acc)
130
131 println("x = ",x_bisection)
132 println("y = ",y_bisection)
133 println("      -                        : ", iters_bisection)
134 println("                             |fx-f| <= ",pogr_bisection)
135
136 xs = a:0.1:b
137 ys = f.(xs)
138
139 Plots.plot(xs, ys, label="f(x)", xlabel="x", ylabel="f(x)", color=:
        purple)
140
141 scatter!([x_bisection], [y_bisection], label="Bisection Min(eps=$acc)",
        color=:red)
142
143 Plots.plot!()
144
145 function golden_section(f, a, b, eps)
146     k = (sqrt(5) - 1) / 2
147     x1 = a + (1 - k) * (b - a)
148     x2 = a + k * (b - a)
149     a = Float64(a)
150     b = Float64(b)
151     iters = 0
152     while abs(x1 - x2) > eps
153         iters += 1
154         if f(x1) <= f(x2)
155             b = x2
156             x2 = x1
157             x1 = a + b - x1
158         else
159             a = x1
160             x1 = x2
161             x2 = a + b - x2
162         end
163     end
```

```
164     min = (a + b) / 2
165     pogr = L*(a-b)/2
166     return min, f(min), iters, pogr
167 end
168
169 using Plots
170
171 a = -10
172 b = -7
173 acc = 0.01
174
175 x_golden, y_golden, iters_golden, pogr_golden = golden_section(f, a, b,
        acc)
176
177 println("x = ",x_golden)
178 println("y = ",y_golden)
179 println("       -                          : ", iters_golden)
180 println("                             |fx-f| <= ",pogr_golden)
181
182 xs = a:0.1:b
183 ys = f.(xs)
184
185 Plots.plot(xs, ys, label="f(x)", xlabel="x", ylabel="f(x)", color=:
        purple)
186
187 scatter!([x_golden], [y_golden], label="Bisection Min(eps=$acc)", color
        =:red)
188
189 Plots.plot!()
190
191 function fibonacci_sequence(n::Int)
192     if n <= 0
193         error("n
                  ")
194     elseif n <= 2
195         return ones(Int, n)
196     end
197
198     seq = Vector{Int}(undef, n)
199     seq[1] = 1
200     seq[2] = 1
201     for i in 3:n
202         seq[i] = seq[i-1] + seq[i-2]
203     end
204     return seq
205 end
```

```
206
207
208 function fibonacci_search(f, a, b, eps)
209     estimated_steps = floor(Int, (b - a) / eps) + 1
210
211     n = 20
212     fib = fibonacci_sequence(n)
213
214     a = Float64(a)
215     b = Float64(b)
216
217     x1 = a + (fib[n-2] / fib[n]) * (b - a)
218     x2 = a + (fib[n-1] / fib[n]) * (b - a)
219
220     iterations = 0
221
222     for k in 1:(n - 3)
223         iterations += 1
224         if f(x1) > f(x2)
225             a = x1
226             x1 = x2
227             x2 = a + (fib[n-k-1] / fib[n-k]) * (b - a)
228         else
229             b = x2
230             x2 = x1
231             x1 = a + (fib[n-k-2] / fib[n-k]) * (b - a)
232         end
233     end
234
235     optimum = (a + b) / 2
236     pogr = L*(a-b) / 2
237     return optimum, f(optimum), iterations, pogr
238 end
239
240 using Plots
241
242 a = -10
243 b = -7
244 acc = 0.01
245
246 x_fibonacci, y_fibonacci, iter_fibonacci, pogr_fibonacci =
        fibonacci_search(f, a, b, acc)
247
248 println("x = ",x_fibonacci)
249 println("y = ",y_fibonacci)
250 println("      -                    : ", iter_fibonacci)
```

```
251
252 println("                              |fx-f| <= ",pogr_fibonacci)
253
254 xs = a:0.1:b
255 ys = f.(xs)
256
257 Plots.plot(xs, ys, label="f(x)", xlabel="x", ylabel="f(x)", color=:
        purple)
258
259 scatter!([x_golden], [y_golden], label="Bisection Min(eps=$acc)", color
        =:red)
260
261 Plots.plot!()
```

3 Результаты

Результаты запуска представлены на рисунках 1 - 4.

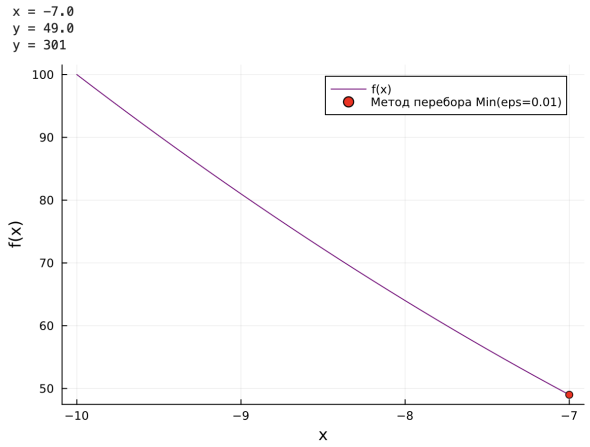

Рис. 1 — Метод перебора

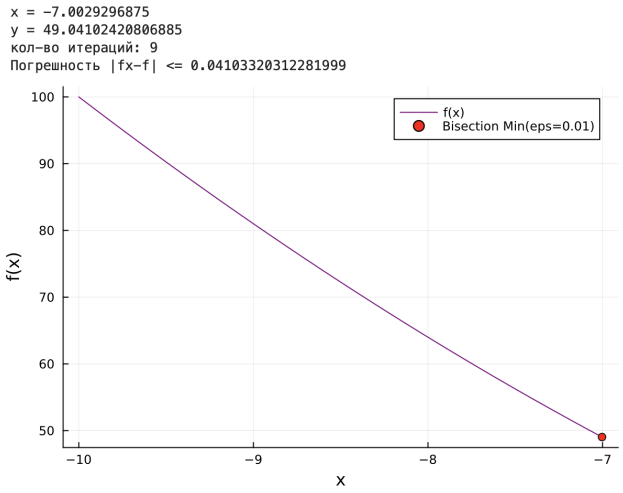

Рис. 2 — Метод дихотомии

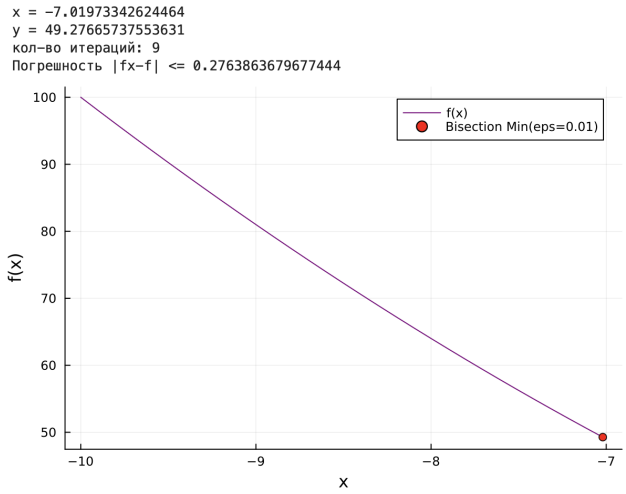

Рис. 3 — Метод золотого сечения

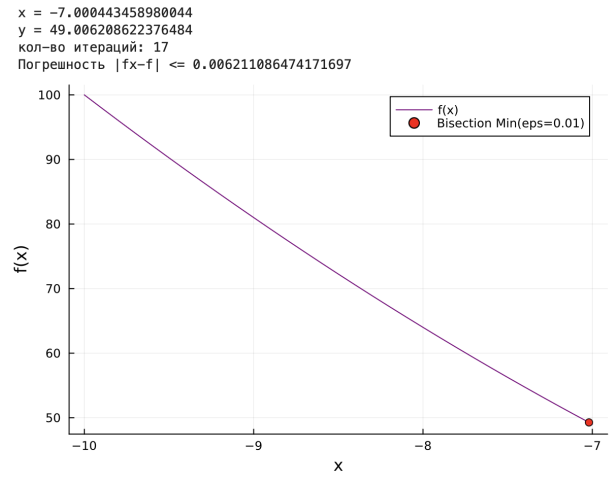

Рис. 4 — Метод фибоначчи

4 Выводы

В ходе выполнения лабораторной работы были реализованы функция проверки унимодальности на отрезке, метод перебора, метод дихотомии (деления пополам), метод фибоначчи, метод золотого сечения для определения минимума функции на заданном отрезке. Результаты работы представлены в виде графиков с выделенными и точками минимума.

Анализируя графики видно, что самым долгим по количеству итераций является метод перебора, наиболее точным и быстрым является метод золотого сечения.