



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 № 5
по курсу «Методы оптимизации»
«Оптимизация различными улучшениями метода градиентного
спуска»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

1. Реализовать метод градиентного спуска для поиска минимума функции.
2. Реализовать метод наискорейшего спуска для поиска минимума функции.
3. Реализовать метод сопряженных градиентов для поиска минимума функции.

2 Реализация

Исходный код программы представлен в листинге 1.

Листинг 1: code

```
1
2 using Plots
3 using LinearAlgebra
4
5 plotly()
6
7
8 function approximate_gradient(f, x, h=1e-6)
9     n = length(x)
10    grad = zeros(n)
11
12    for i in 1:n
13        x_plus_h = copy(x)
14        x_plus_h[i] += h
15
16        x_minus_h = copy(x)
17        x_minus_h[i] -= h
18
19        grad[i] = (f(x_plus_h) - f(x_minus_h)) / (2*h)
20    end
21
22    return grad
23 end
24
25 function gradient_descent(f, x0, max_iterations=1000000, tol=1e-6)
26     x = copy(x0)
27     xs = [copy(x0)]
28
29     learning_rate = 0.01
30
31     for i in 1:max_iterations
```

```

32         grad = approximate_gradient(f, x)
33
34         if norm(grad) < tol
35             break
36         end
37
38         x_new = x - learning_rate * grad
39
40         if f(x_new) < f(x)
41             x = x_new
42             learning_rate *= 1.05
43         else
44             learning_rate *= 0.5
45             x_new = x - learning_rate * grad
46             x = x_new
47         end
48
49         push!(xs, copy(x))
50
51         if i > 1 && norm(xs[end] - xs[end-1]) < tol
52             break
53         end
54     end
55
56     return x, xs
57 end
58
59
60 function golden_section(f, a, b)
61     k = (sqrt(5) - 1) / 2
62     x1 = a + (1 - k) * (b - a)
63     x2 = a + k * (b - a)
64
65     a = Float64(a)
66     b = Float64(b)
67
68     while abs(x1 - x2) > 1e-6
69         if f(x1) <= f(x2)
70             b = x2
71             x2 = x1
72             x1 = a + b - x1
73         else
74             a = x1
75             x1 = x2
76             x2 = a + b - x2
77         end

```

```

78     end
79     return (a + b) / 2
80 end
81
82 function steepest_descent(f, x0, max_iterations=100000, tol=1e-6)
83     x = copy(x0)
84     xs = [copy(x0)]
85
86     for i in 1:max_iterations
87         grad = approximate_gradient(f, x)
88
89         if norm(grad) < tol
90             break
91         end
92
93         #
94         line_search_function( ) = f(x -      * grad)
95
96         result = golden_section(line_search_function, 0.0, 10.0)
97         = result
98
99         x_new = x -      * grad
100
101         if f(x_new) < f(x)
102             x = x_new
103         end
104
105         push!(xs, copy(x))
106
107         if i > 1 && norm(xs[end] - xs[end-1]) < tol
108             break
109         end
110     end
111
112     return x, xs
113 end
114
115 a=1
116 b=2
117 f(x) = a*x[1]^2 + b * x[2]^2
118 x0 = [10.0, 10.0]
119
120 println(" ")
121 r_grad,xs = gradient_descent(f,x0)
122 println(" : ", r_grad)
123 # println(" : ", length(xs))

```

```

124 x_coords_grad = [x[1] for x in xs]
125 y_coords_grad = [y[2] for y in xs]
126
127 r_step, xs = steepest_descent(f, x0)
128 println("                                : ",
        r_step)
129 # println("                                :
        ", length(xs))
130 x_coords_grad_fast = [x[1] for x in xs]
131 y_coords_grad_fast = [y[2] for y in xs]
132
133 r_conj, xs = conjugate_gradient(f, x0)
134 println("                                : ",
        r_conj)
135 # println("                                : ",
        length(xs))
136 x_coords_grad_conj = [x[1] for x in xs]
137 y_coords_grad_conj = [y[2] for y in xs]
138
139 println()
140
141 x = -10:0.1:10
142 y = -10:0.1:10
143
144 plt1 = contour(x, y, (x, y) -> f([x, y]),
145     color=:thermal,
146     levels=20,
147     xlabel=" x ",
148     ylabel=" y ",
149     title="(2D) ",
150     legend=:topleft,
151     size=(700, 500)
152 )
153
154 scatter!(plt1,
155     [x0[1]],
156     [x0[2]],
157     color=:green,
158     markersize=8,
159     label=" "
160 )
161
162 scatter!(plt1,
163     [r_grad[1]],
164     [r_grad[2]],
165     color=:blue,

```

```

166     markersize=8,
167     label="                "
168 )
169
170 scatter!(plt1,
171     [r_step[1]],
172     [r_step[2]],
173     color=:red,
174     markersize=8,
175     label="                "
176 )
177
178 scatter!(plt1,
179     [r_conj[1]],
180     [r_conj[2]],
181     color=:yellow,
182     markersize=8,
183     label="                "
184 )
185
186
187 plt1 = plot!(x_coords_grad, y_coords_grad, label="
                ", line=:blue, color=:blue, marker=:circle, markersize
                =4)
188 plt1 = plot!(x_coords_grad_fast, y_coords_grad_fast, label="
                ", line=:
                red, color=:red, marker=:circle, markersize=4)
189 plt1 = plot!(x_coords_grad_conj, y_coords_grad_conj, label="
                ", line=:yellow,
                color=:yellow, marker=:circle, markersize=4)
190
191
192 display(plt1)

```

3 Результаты

Результаты запуска представлены на рисунках 1.

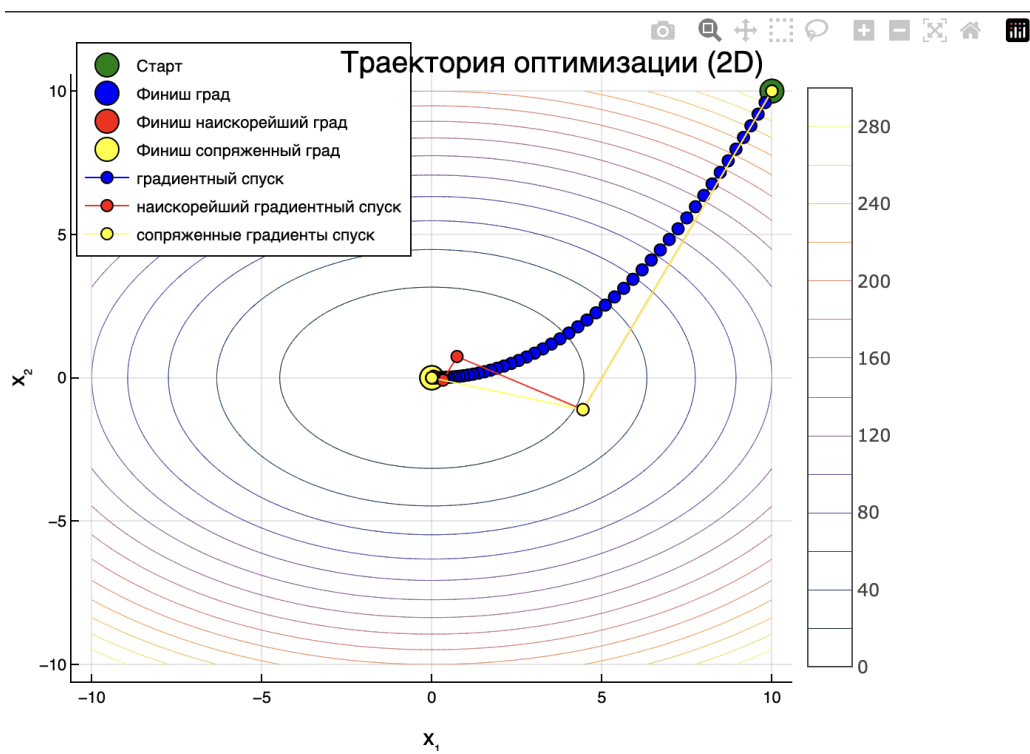


Рис. 1 — Визуализация методов

4 Выводы

Все три метода успешно нашли минимум целевой функции, однако скорость и эффективность сходимости различались в зависимости от особенностей функции. Выбор метода оптимизации должен основываться на структуре целевой функции и доступных вычислительных ресурсах. Например, если размерность задачи невелика, а целевая функция слабо искривлена, можно использовать метод градиентного спуска. Для более сложных функций целесообразно применять метод наискорейшего спуска или метод сопряженных градиентов.

Визуализация траекторий оптимизации подтвердила, что метод сопряженных градиентов требует меньшего количества итераций для достижения заданной точности по сравнению с другими методами.

Таким образом, в ходе выполнения лабораторной работы мы убедились в преимуществах и недостатках различных методов оптимизации и научились выбирать наиболее подходящий метод в зависимости от задачи.