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 № 8

по курсу «Методы оптимизации»

«Оптимизация квазиньютоновскими методами»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

1. Реализовать метод BFGS (Бройдена, Флэтчера, Гольдфарба, Шанно).
2. Реализовать метод Дэвидона-Флэтчера-Пауэла

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```
1
2  using Plots
3  using LinearAlgebra
4
5  plotly()
6
7  function approximate_gradient(f, x, h=1e-8)
8      n = length(x)
9      grad = zeros(n)
10
11     for i in 1:n
12         x_plus_h = copy(x)
13         x_plus_h[i] += h
14
15         x_minus_h = copy(x)
16         x_minus_h[i] -= h
17
18         grad[i] = (f(x_plus_h) - f(x_minus_h)) / (2*h)
19     end
20
21     return grad
22 end
23
24 function golden_section_search(f, a, b, tol=1e-6, max_iter=100)
25     golden_ratio = (sqrt(5) + 1) / 2
26     c = b - (b - a) / golden_ratio
27     d = a + (b - a) / golden_ratio
28
29     fc = f(c)
30     fd = f(d)
31
32     iter = 0
33     while abs(b - a) > tol && iter < max_iter
34         iter += 1
```

```
35
36          if fc < fd
37              b = d
38              d = c
39              fd = fc
40              c = b - (b - a) / golden_ratio
41              fc = f(c)
42          else
43              a = c
44              c = d
45              fc = fd
46              d = a + (b - a) / golden_ratio
47              fd = f(d)
48          end
49      end
50
51      return (a + b) / 2
52  end
53
54  function swann_method(f, x0, h=0.1)
55      first = x0
56      second = x0 + h
57      if f(second) > f(first)
58          h = -h
59          first, second = second, second + h
60      end
61      last = second + h
62
63      while f(last) < f(second)
64          h *= 2
65          first, second, last = second, last, last + h
66      end
67      if second > last
68          first, second, last = last, second, first
69      end
70
71      return first, last
72  end
73
74  function bfgs(f, x0; tol=1e-4, max_iter=10000)
75      x = x0
76      n = length(x)
77      H = I(n)
78      println(H)
79      trajectory = [x]
80
```

```
81      for _ in 1:max_iter
82          g = approximate_gradient(f, x)
83          if norm(g) < tol
84              break
85          end
86
87          p = -H * g
88          ff(alpha) = f(x + alpha * p)
89          a, b = swann_method(ff, 0.0)
90          alpha = golden_section_search(ff, a, b)
91
92          x_new = x + alpha * p
93          delta_x = x_new - x
94          delta_g = approximate_gradient(f, x_new) - g
95
96          if dot(delta_x, delta_g) > 0
97              rho = 1.0 / dot(delta_x, delta_g)
98              I_n = I(n)
99              H = (I_n - rho * delta_x * delta_g') * H * (I_n - rho *
    delta_g * delta_x') + rho * delta_x * delta_x'
100             println(H)
101         end
102
103         x = x_new
104         push!(trajectory, x)
105     end
106
107     return x, trajectory
108 end
109
110 function dfp(f, x0; tol=1e-4, max_iter=1000)
111     x = x0
112     n = length(x)
113     H = I(n)  #                                    H -
114     println(H)
115     trajectory = [x]  #

116
117     for iter in 1:max_iter
118         g = approximate_gradient(f,x)
119         d = -H * g  #
120
121         ff(alpha) = f(x + alpha * d)
122         a, b = swann_method(ff, 0.0)
123         alpha = golden_section_search(ff, a, b)
124
```

```
125         x_new = x + alpha * d
126         g_new = approximate_gradient(f,x_new)
127 
128         s = x_new - x
129         y = g_new - g
130 
131         #                                          H
132         ys = y' * s
133         H = H + (s * s') / ys - (H * (y * y') * H) / (y' * H * y)
134         println(H)
135 
136         #
137         push!(trajectory, x_new)
138 
139         #
140         if norm(g_new) < tol
141             return x_new, trajectory  #
142         end
143 
144         x = x_new
145     end
146 
147     return x, trajectory  #
148 end
149 
150 function plot_optimization_paths(f, x_range, y_range, paths_with_names,
        title="                                                    ",
        global_min=nothing)
151     z = [f([x, y]) for y in y_range, x in x_range]
152 
153     clamp_level = maximum(filter(isfinite, z)) / 2
154     z_clamped = [min(val, clamp_level) for val in z]
155 
156     p = contour(x_range, y_range, z_clamped,
157                 fill=false,
158                 levels=20,
159                 color=:thermal,
160                 xlabel="x",
161                 ylabel="y",
162                 title=title,
163                 size=(800, 600))
164 
165     colors = [:red, :green, :blue, :purple, :orange]
```

```
166     for (i, (name, path)) in enumerate(paths_with_names)
167         x_coords = [point[1] for point in path]
168         y_coords = [point[2] for point in path]
169 
170         plot!(p, x_coords, y_coords,
171               label=name,
172               line=(colors[i], 2),
173               marker=(:circle, 2, 0.5))
174 
175         annotate!(p, x_coords[1], y_coords[1], text("          ", :left,
      8, :white))
176         annotate!(p, x_coords[end], y_coords[end], text("          ", :
     right, 8, :white))
177     end
178 
179     if global_min !== nothing
180         scatter!(p, [global_min[1]], [global_min[2]],
181                  label="                                        ",
182                  color=:white,
183                  markersize=5,
184                  markerstrokewidth=1,
185                  markerstrokecolor=:black)
186     end
187 
188     return p
189 end
190 
191 
192 function plot_optimization_paths(f, x_range, y_range, paths_with_names,
     title="                                                          ",
     global_min=nothing)
193     z = [f([x, y]) for y in y_range, x in x_range]
194 
195     clamp_level = maximum(filter(isfinite, z)) / 2
196     z_clamped = [min(val, clamp_level) for val in z]
197 
198     p = contour(x_range, y_range, z_clamped,
199                 fill=false,
200                 levels=20,
201                 color=:thermal,
202                 xlabel="x",
203                 ylabel="y",
204                 title=title,
205                 size=(800, 600))
206 
207     colors = [:red, :green, :blue, :purple, :orange]
```

```
208     for (i, (name, path)) in enumerate(paths_with_names)
209         x_coords = [point[1] for point in path]
210         y_coords = [point[2] for point in path]
211 
212         plot!(p, x_coords, y_coords,
213               label=name,
214               line=(colors[i], 2),
215               marker=(:circle, 2, 0.5))
216 
217         annotate!(p, x_coords[1], y_coords[1], text("          ", :left,
      8, :white))
218         annotate!(p, x_coords[end], y_coords[end], text("          ", :
     right, 8, :white))
219     end
220 
221     if global_min !== nothing
222         scatter!(p, [global_min[1]], [global_min[2]],
223                  label="                                        ",
224                  color=:white,
225                  markersize=5,
226                  markerstrokewidth=1,
227                  markerstrokecolor=:black)
228     end
229 
230     return p
231 end
232 
233 # 1.
234 function rosenbrock(x)
235     return (1.0 - x[1])^2 + 100.0*(x[2] - x[1]^2)^2
236 end
237 
238 # 2.
239 function rastrigin(x)
240     return 20 + x[1]^2 - 10*cos(2*pi*x[1]) + x[2]^2 - 10*cos(2*pi*x[2])
241 end
242 
243 # 3.
244 function schwefel(x)
245     return 418.9829*2 - (x[1]*sin(sqrt(abs(x[1]))) + x[2]*sin(sqrt(abs(x
     [2]))))
246 end
247 
248 function my(x)
249     return (x[1] - 4*x[2])^2 + (x[2] + 5)^2
250 end
```

```
251
252 a = 7.0
253 b = 5.0
254 function quadratic(x)
255     return a * x[1]^2 + b * x[2]^2
256 end
257
258
259 function run_optimization(f, x0, title, x_range, y_range, global_min=
        nothing)
260     println("\n======== $(title) ========")
261
262     result_bfgs, path_bfgs = bfgs(f, x0)
263
264     println("           BFGS (                  ,                  ,
                           ,           ) (                  ): ",
        result_bfgs)
265     println("                                  : ", f(result_bfgs))
266     println("                                      : ", length(path_bfgs)
        )
267     println()
268
269     result_dfp, path_dfp = dfp(f, x0)
270     println("           DFP (                  ): ", result_dfp)
271     println("                                  : ", f(result_dfp))
272     println("                                      : ", length(path_dfp))
273     println()
274
275     paths = [
276         ("BFGS", path_bfgs),
277         ("DFP",path_dfp),
278     ]
279
280     p = plot_optimization_paths(f, x_range, y_range, paths, title,
        global_min)
281
282     display(p)
283
284     return nothing
285 end
286
287 #
288 x0_rosenbrock = [-1.2, 2.0]
289 x_range_rosenbrock = -3.0:0.1:3.0
290 y_range_rosenbrock = -1.0:0.1:4.0
291 global_min_rosenbrock = [1.0, 1.0]
```

```
292 run_optimization(rosenbrock, x0_rosenbrock, "
                            ", x_range_rosenbrock, y_range_rosenbrock,
    global_min_rosenbrock)
```

3 Результаты

Результаты запуска представлены на рисунках 1.

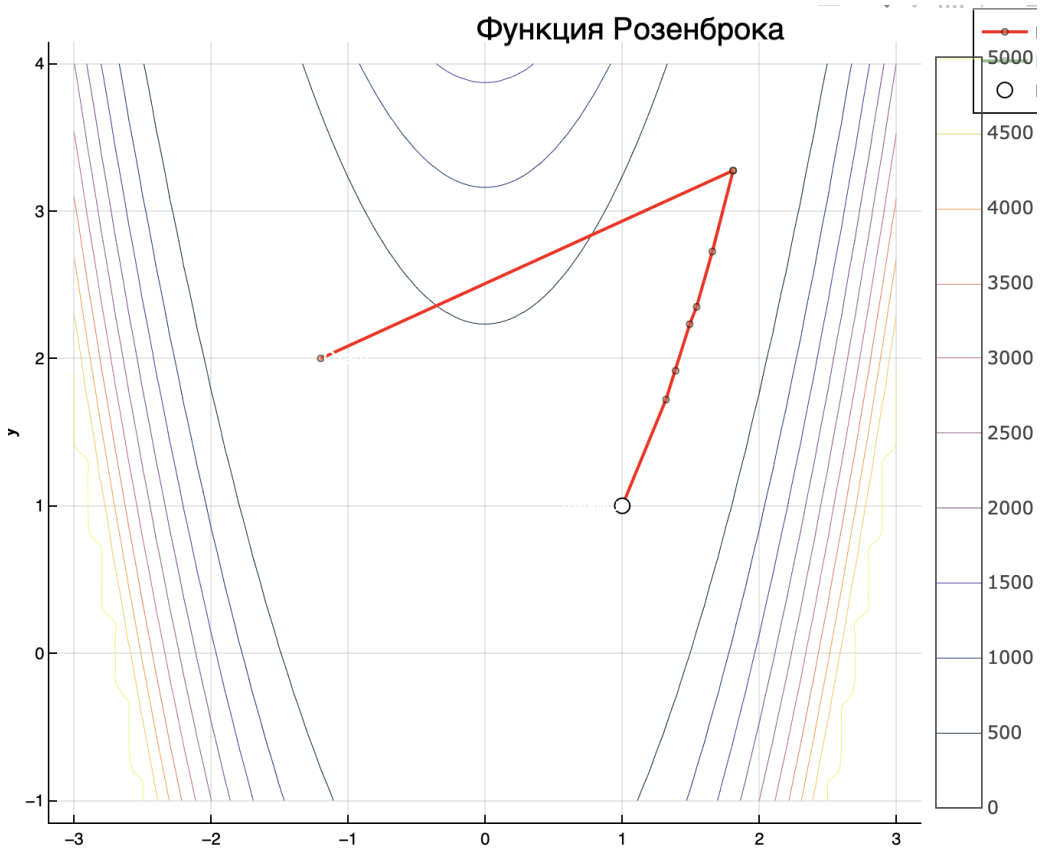

Рис. 1 — Визуализация методов 1

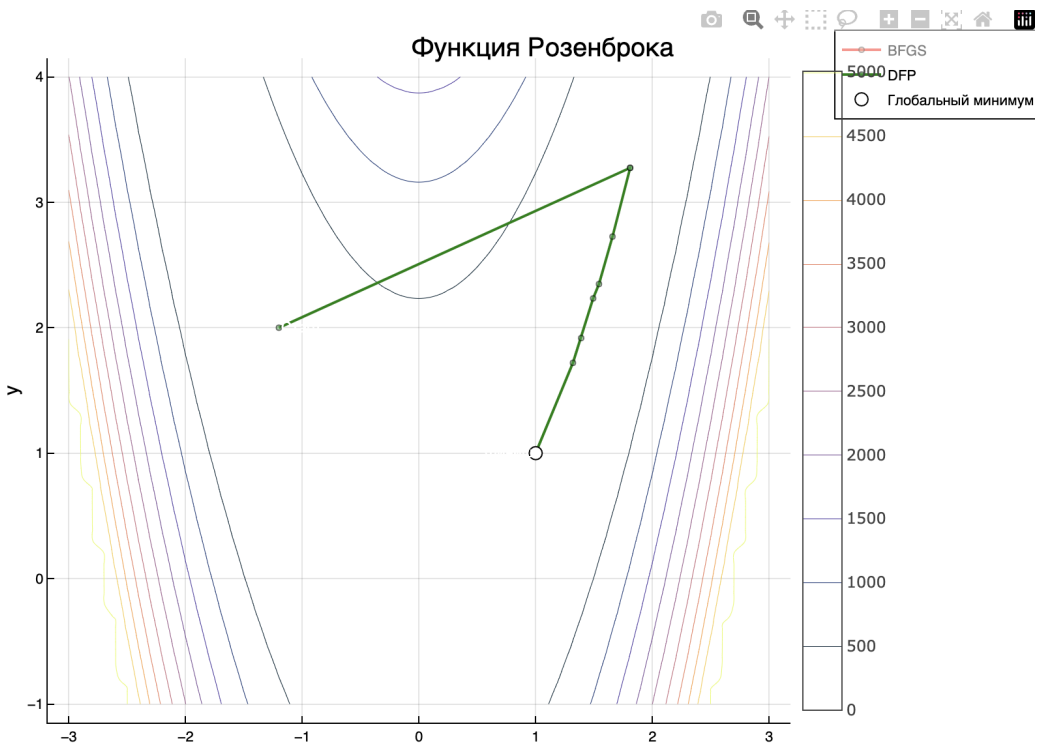

Рис. 2 — Визуализация методов 2

4 Выводы

В результате данной лабораторной работы были реализованы квазиньютоновские методы оптимизации, которые аппроксимируют матрицу Гессе, позволяя не вычислять вторые производные. Методы показали отличную сходимость на овражных функциях.