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 № 2

по курсу «Методы оптимизации»

«Численные методы минимизации функции нескольких переменных»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

1. Реализовать метод покоординатного спуска.
2. Реализовать метод Гусса-Зейделя.
3. Реализовать метод Хука-Дживса.

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```
1
2  using Plots
3  using LinearAlgebra
4
5  #                                    (                    )
6  f(x) = (1.0 - x[1])^2 + 100.0*(x[2] - x[1]^2)^2
7  # f(x) = x[1]^2+x[2]^2 + 2*x[1]*x[2]
8
9  #
10 x0 = [-2.0, -2.0]
11
12 function coordinate_descent1(f, x0, eps1=1e-6, eps2=1e-10)
13     x = x0
14     n = length(x)
15     trajectory = [x]
16     alpha = 0.1
17
18     while true
19         x_new = copy(x)
20         min_f = f(x)
21         changed = false
22         while true
23             for i in 1:n
24                 for j in [-1, 1]
25                     xi = copy(x)
26                     xi[i] += alpha * j
27                     if f(xi) <= min_f
28                         x_new = xi
29                         min_f = f(xi)
30                         changed = true
31                     end
32                 end
```

```
            end
            if changed
                break
            end
            alpha /= 2
        end

        if norm(x - x_new) < eps1 || norm(f(x) - f(x_new)) < eps2
            break
        end

        x = copy(x_new)
        push!(trajectory, x)
    end

    return x, trajectory
end


result_coord, history_coord = coordinate_descent1(f, x0)

println(result_coord)
println("      -                        :",length(history_coord))

#                                              3D
x = range(-2.0, 2.0, length=100)
y = range(-2.0, 2.0, length=100)
X = repeat(reshape(x, 1, :), length(y), 1)
Y = repeat(y, 1, length(x))
Z = map((a,b) -> f([a,b]), X, Y)

#                    3D
plt = surface(x, y, Z, color=:thermal, alpha=0.5, legend=true)
# scatter3d!(plt,
#     [p[1] for p in history_coord],
#     [p[2] for p in history_coord],
#     [f(p) for p in history_coord],
#     markersize=3,
#     color=:red,
#     label="Search Path"
# )

#
plot!(plt,
    [p[1] for p in history_coord],
    [p[2] for p in history_coord],
```

```
    [f(p) for p in history_coord],
    linecolor=:blue,
    label="Connecting Lines"
)

scatter3d!(plt, [x0[1]], [x0[2]], [f(x0)], color=:green, label="Start")
scatter3d!(plt, [result_coord[1]], [result_coord[2]], [f(result_coord)],
    color=:blue, label="End")

title!(plt, "
                                                    ")
xlabel!(plt, "x")
ylabel!(plt, "y")
zlabel!(plt, "f(x,y)")

#
plt = contour(x, y, Z,
    color=:thermal,
    levels=20,
    xlabel=" x  ",
    ylabel=" x  ",
    title="                                                (2D)",
    legend=:topleft
)

#
threshold = 0.1
filtered_x = []
filtered_y = []

last_x = nothing
last_y = nothing

for p in history_coord
    if last_x === nothing || abs(p[1] - last_x) > threshold || abs(p[2]
    - last_y) > threshold
        push!(filtered_x, p[1])
        push!(filtered_y, p[2])
        last_x = p[1]
        last_y = p[2]
    end
end

plot!(plt,
    filtered_x,
    filtered_y,
```

```
    color=:red,
    linewidth=1,
    label="                                    ",
    marker=:circle,
    markersize=4
)

print(x0[1],x0[2])

#
scatter!(plt,
    [x0[1], result_coord[1]],
    [x0[2], result_coord[2]],
    color=[:green :blue],
    markersize=8,
    label=["          " "          "]
)

#
plot!(plt, colorbar=true)

#

annotate!(plt,
    [(x0[1], x0[2], text("f = $(round(f(x0), digits=3))", 8, :left)),
    (result_coord[1], result_coord[2], text("f = $(round(f(result_coord)
    , digits=3))", 8, :right))]
)

using Plots
using LinearAlgebra

function swann_method(f, x0, h=0.1)
    first = x0
    second = x0 + h
    if f(second) > f(first)
        h = -h
        first, second = second, second + h
    end
    last = second + h

    while f(last) < f(second)
        h *= 2
        first, second, last = second, last, last + h
    end
    if second > last
```

```
166             first, second, last = last, second, first
167         end
168
169         return first, last
170     end
171
172     function golden_section(f, a, b)
173         k = (sqrt(5) - 1) / 2
174         x1 = a + (1 - k) * (b - a)
175         x2 = a + k * (b - a)
176
177         a = Float64(a)
178         b = Float64(b)
179
180         while abs(x1 - x2) > 1e-6
181             if f(x1) <= f(x2)
182                 b = x2
183                 x2 = x1
184                 x1 = a + b - x1
185             else
186                 a = x1
187                 x1 = x2
188                 x2 = a + b - x2
189             end
190         end
191         return (a + b) / 2
192     end
193
194     function gauss_zeidel(f, x0, eps1=1e-6, eps2=1e-10)
195         x = x0
196         n = length(x)
197         trajectory = [x]
198
199         while true
200             x_new = copy(x)
201             min_f = f(x)
202             for i in 1:n
203                 for j in [-1, 1]
204                     function g(alpha)
205                         xi = copy(x)
206                         xi[i] += alpha * j
207                         return f(xi)
208                     end
209                     l, r = swann_method(g, 0.5)
210                     alpha = golden_section(g, l, r)
211                     xi = copy(x)
```

```
                xi[i] += alpha * j
                if f(xi) < min_f
                    x_new = xi
                    min_f = f(xi)
                end
            end
        end

        if norm(x - x_new) < eps1 || norm(f(x) - f(x_new)) < eps2
            break
        end

        x = copy(x_new)
        push!(trajectory, x)
    end

    return x, trajectory, length(trajectory)
end

result_gauss, history_gauss,iters_count_gauss = gauss_zeidel(f, x0)

println(result_gauss)
println("      -                        :",iters_count_gauss)

#                                              3D
x = range(-2.0, 2.0, length=100)
y = range(-2.0, 2.0, length=100)
X = repeat(reshape(x, 1, :), length(y), 1)
Y = repeat(y, 1, length(x))
Z = map((a,b) -> f([a,b]), X, Y)

#                   3D
plt = surface(x, y, Z, color=:thermal, alpha=0.5, legend=true)
# scatter3d!(plt,
#     [p[1] for p in history_gauss],
#     [p[2] for p in history_gauss],
#     [f(p) for p in history_gauss],
#     markersize=3,
#     color=:red,
#     label="Search Path"
# )

#
plot!(plt,
    [p[1] for p in history_gauss],
    [p[2] for p in history_gauss],
```

```
    [f(p) for p in history_gauss],
    linecolor=:blue,
    label="Connecting Lines"
)
scatter3d!(plt, [x0[1]], [x0[2]], [f(x0)], color=:green, label="Start")
scatter3d!(plt, [result_gauss[1]], [result_gauss[2]], [f(result_gauss)],
    color=:blue, label="End")

title!(plt, "                                    -              ")
xlabel!(plt, "x")
ylabel!(plt, "y")
zlabel!(plt, "f(x,y)")

using Plots

#
plt = contour(x, y, Z,
    color=:thermal,
    levels=20,
    xlabel=" x  ",
    ylabel=" x  ",
    title="                                          (2D)",
    legend=:topleft
)

threshold = 0.1
filtered_x = []
filtered_y = []

last_x = nothing
last_y = nothing

for p in history_gauss
    if last_x === nothing || abs(p[1] - last_x) > threshold || abs(p[2]
    - last_y) > threshold
        push!(filtered_x, p[1])
        push!(filtered_y, p[2])
        last_x = p[1]
        last_y = p[2]
    end
end

plot!(plt,
    filtered_x,
    filtered_y,
    color=:red,
```

```
    linewidth=1,
    label="                                    ",
    marker=:circle,
    markersize=4
)

print(x0[1],x0[2])

#
scatter!(plt,
    [x0[1], result_gauss[1]],
    [x0[2], result_gauss[2]],
    color=[:green :blue],
    markersize=8,
    label=["          " "          "]
)

#
plot!(plt, colorbar=true)

#

annotate!(plt,
    [(x0[1], x0[2], text("f = $(round(f(x0), digits=3))", 8, :left)),
    (result_gauss[1], result_gauss[2], text("f = $(round(f(result_gauss)
    , digits=3))", 8, :right))]
)

function exploratory_search(f, x, delta)
    n = length(x)
    x_new = copy(x)
    for i in 1:n
        start_val = f(x_new)
        temp = x_new[i]
        x_new[i] = temp + delta[i]
        f_plus = f(x_new)
        if f_plus >= start_val
            x_new[i] = temp - delta[i]
            f_minus = f(x_new)
            if f_minus >= start_val
                x_new[i] = temp
            end
        end
    end
    return x_new
end
```

```
346
347 function hooke_jeeves(f, x0, eps=1e-6, delta0=0.1)
348     n = length(x0)
349     delta = fill(delta0, n)
350     x = copy(x0)
351     trajectory = [x]
352
353     while maximum(delta) > eps
354         x_exp = exploratory_search(f, x, delta)
355
356         if f(x_exp) >= f(x)
357             delta = delta / 2
358         end
359         while f(x_exp) < f(x)
360             direction = x_exp - x
361             x = copy(x_exp)
362             push!(trajectory, x)
363             x_pattern = x + direction
364             x_exp = exploratory_search(f, x_pattern, delta)
365         end
366     end
367
368     return x, trajectory
369 end
370
371 result_hooke, history_hooke = hooke_jeeves(f, x0)
372 println(result_hooke)
373 println("      -                    :",length(history_hooke))
374
375 #                                          3D
376 x = range(-2.0, 2.0, length=100)
377 y = range(-2.0, 2.0, length=100)
378 X = repeat(reshape(x, 1, :), length(y), 1)
379 Y = repeat(y, 1, length(x))
380 Z = map((a,b) -> f([a,b]), X, Y)
381
382 #                  3D
383 plt = surface(x, y, Z, color=:thermal, alpha=0.5, legend=true)
384 # scatter3d!(plt,
385 #     [p[1] for p in history_hooke],
386 #     [p[2] for p in history_hooke],
387 #     [f(p) for p in history_hooke],
388 #     markersize=3,
389 #     color=:red,
390 #     label="Search Path"
391 # )
```

```
392
393 #
394 plot!(plt,
395     [p[1] for p in history_hooke],
396     [p[2] for p in history_hooke],
397     [f(p) for p in history_hooke],
398     linecolor=:blue,
399     label="Connecting Lines"
400 )
401
402 scatter3d!(plt, [x0[1]], [x0[2]], [f(x0)], color=:green, label="Start")
403 scatter3d!(plt, [result_hooke[1]], [result_hooke[2]], [f(result_hooke)],
        color=:blue, label="End")
404
405 title!(plt, "                                                    -
    ")
406 xlabel!(plt, "x")
407 ylabel!(plt, "y")
408 zlabel!(plt, "f(x,y)")
409
410 using Plots
411
412 #
413 plt = contour(x, y, Z,
414     color=:thermal,
415     levels=20,
416     xlabel=" x  ",
417     ylabel=" x  ",
418     title="                                              (2D)",
419     legend=:topleft
420 )
421
422 #
423 threshold = 0.1
424 filtered_x = []
425 filtered_y = []
426
427 last_x = nothing
428 last_y = nothing
429
430 for p in history_hooke
431     if last_x === nothing || abs(p[1] - last_x) > threshold || abs(p[2]
    - last_y) > threshold
432         push!(filtered_x, p[1])
433         push!(filtered_y, p[2])
434         last_x = p[1]
```

```
435             last_y = p[2]
436         end
437 end
438
439 plot!(plt,
440     filtered_x,
441     filtered_y,
442     color=:red,
443     linewidth=1,
444     label="                                    ",
445     marker=:circle,
446     markersize=4
447 )
448
449 #
450 scatter!(plt,
451     [x0[1], result_hooke[1]],
452     [x0[2], result_hooke[2]],
453     color=[:green :blue],
454     markersize=8,
455     label=["          " "          "]
456 )
457
458 #
459 plot!(plt, colorbar=true)
460
461 #

462 annotate!(plt,
463     [(x0[1], x0[2], text("f = $(round(f(x0), digits=3))", 8, :left)),
464     (result_hooke[1], result_hooke[2], text("f = $(round(f(result_gauss)
        , digits=3))", 8, :right))]
465 )
```

3 Результаты

Результаты запуска представлены на рисунках 1 - 6.

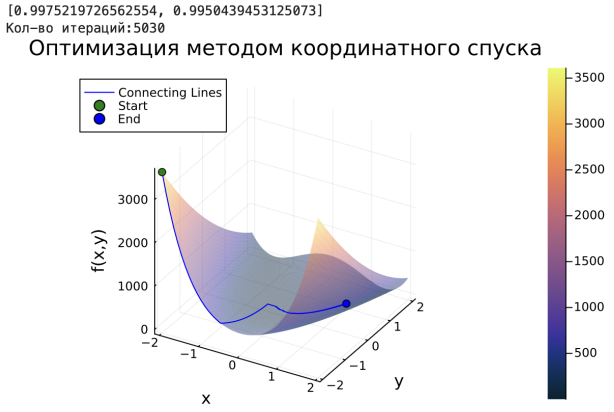

Рис. 1 — Метод покоординатного спуска 3D

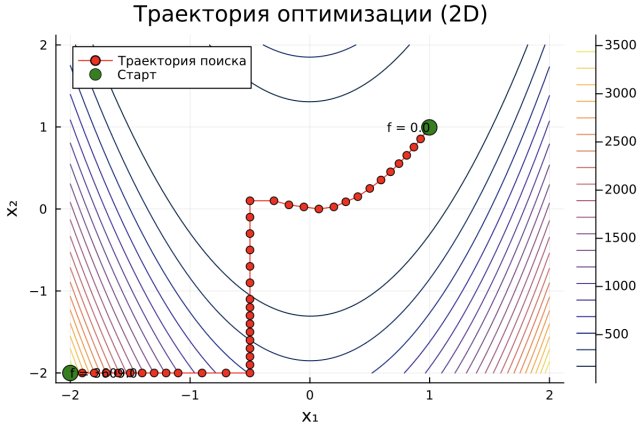

Рис. 2 — Метод покоординатного спуска 2D

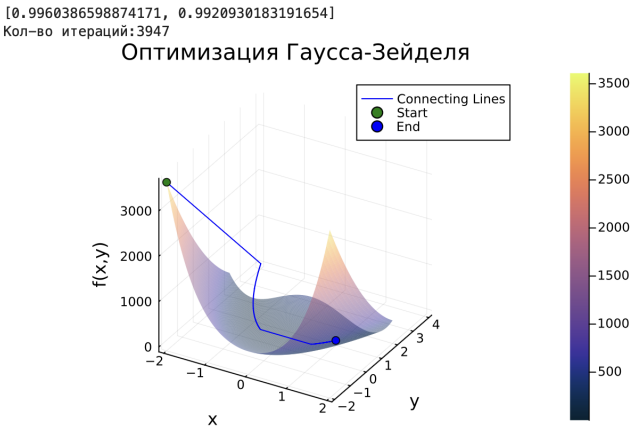

Рис. 3 — Метод Гаусса-Зейделя 3D

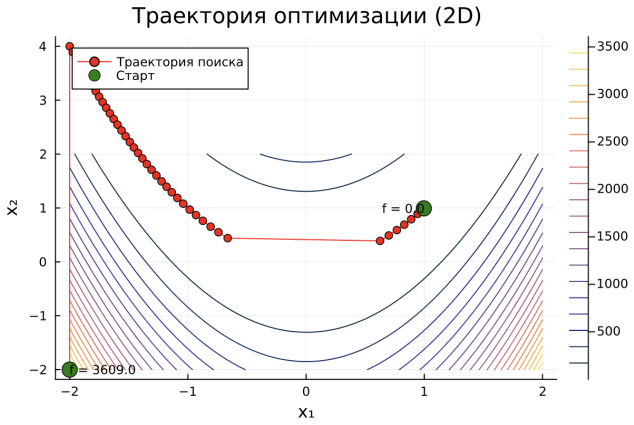

Рис. 4 — Метод Гаусса-Зейделя 2D

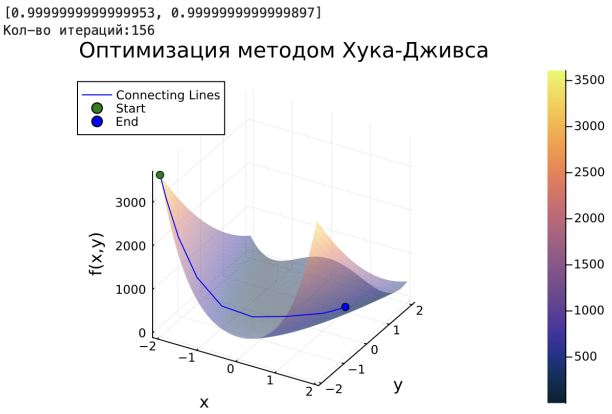

Рис. 5 — Метод Хука-Дживса 3D

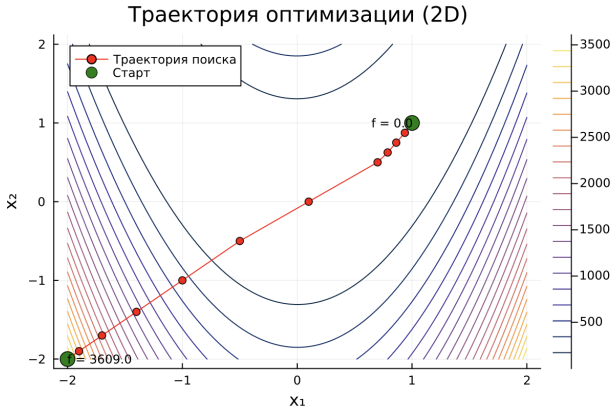

Рис. 6 — Метод Хука-Дживса 2D

4 Выводы

В ходе выполнения лабораторной работы были реализованы различные методы поиска безусловного минимума функций. На графиках была наглядна продемонстрирована работа этих методов. Для первых двух методов мы кажды

раз продвигаемся по 1 координате, тогда как в последнем можем совершать шаги вдоль нескольких координат, тем самым наблюдая лучшую сходимость на овражных функциях.