Thuật toán solve\_lagrange(vars, lambda)

Đầu vào:

vars - Vector các biến (x1, x2) ban đầu

lambda - Giá trị giả thuyết ban đầu của nhân tử Lagrange

Đầu ra:

Giá trị tối ưu của các biến và nhân tử Lagrange (nếu tồn tại)

Biến:

tolerance - Ngưỡng sai số cho hội tụ

max\_iterations - Số lần lặp tối đa

grad - Gradient của hàm Lagrange

jacobian - Ma trận Jacobian của gradient

delta - Bước cập nhật cho các biến

1. Đặt tolerance = 1e-6

2. Đặt max\_iterations = 1000

3. Khởi tạo vars = [2, 2] (giá trị giả định ban đầu)

4. Khởi tạo lambda = 0.5 (giá trị giả định ban đầu)

5. Lặp qua tối đa max\_iterations lần:

a. Tính gradient: grad = lagrange\_grad(vars, lambda)

b. Tính Jacobian: jacobian = lagrange\_jacobian(vars, lambda)

c. Nếu norm(grad) < tolerance thì:

- Kết luận: Hội tụ, thoát khỏi vòng lặp

d. Giải phương trình: jacobian \* delta = -grad

(Tìm delta bằng phương pháp giải hệ tuyến tính)

e. Cập nhật các biến:

vars[0] = vars[0] + delta[0]

vars[1] = vars[1] + delta[1]

lambda = lambda + delta[2]

f. In thông tin:

"Iteration i: x1 = vars[0], x2 = vars[1], lambda = lambda"

g. Kiểm tra điều kiện KKT:

Nếu check\_KKT\_conditions(vars, lambda) thì:

- In "Điều kiện KKT thỏa mãn tại bước lặp thứ i"

- Thoát khỏi vòng lặp

Nếu lambda < 0 thì:

- In "Không thỏa điều kiện KKT"

- Trả về false

6. Nếu vòng lặp hoàn tất mà không hội tụ:

a. In "Nhập lại giá trị x1, x2, lambda"

b. Trả về false

7. Nếu hội tụ:

a. In "Optimal point: x1 = vars[0], x2 = vars[1]"

b. In "Maximum value of f(x1, x2): objective(vars)"

c. In "Lagrange Multiplier (lambda): lambda"

d. Trả về true

Kết thúc thuật toán

Hàm GOLDEN\_SECTION(func, a, b, tol):

gr = (sqrt(5) + 1) / 2 // Tỷ lệ vàng

c = b - (b - a) / gr

d = a + (b - a) / gr

Lặp cho đến khi |c - d| < tol:

Nếu func(c) < func(d):

b = d

Ngược lại:

a = c

Cập nhật c và d dựa trên tỷ lệ vàng

Trả về (b + a) / 2

Hàm POWELL\_METHOD(func, iniGuess, tol, maxIter):

Khởi tạo X = iniGuess

Khởi tạo Y = X

Khởi tạo directions = [[1, 0], [0, 1]] // Các hướng tìm kiếm ban đầu

count = 1

doLoop = true

Lặp khi doLoop và count <= maxIter:

Cho mỗi hướng q trong directions:

Xác định hàm func\_gS(alpha) là giá trị của func tại (X[0] + alpha \* directions[q][0], X[1] + alpha \* directions[q][1])

Tìm alphaOpti bằng GOLDEN\_SECTION(func\_gS, searchRange[0], searchRange[1])

Cập nhật X theo alphaOpti và directions[q]

Nếu q là hướng cuối cùng:

Tăng count

Tạo hướng mới newDir từ chênh lệch giữa X và Y

Chuẩn hóa newDir và thêm vào directions

Loại bỏ hướng đầu tiên khỏi directions

Cập nhật Y = X

Nếu khoảng cách giữa Y và X < tol:

doLoop = false

Nếu count đạt maxIter:

Đặt lại directions và count

Kết thúc vòng lặp

Nếu vượt quá maxIter:

Trả về rỗng (không tìm thấy nghiệm)

Trả về X

Hàm AUGMENTED\_LAGRANGIAN(Ffunc, gFuncs, HFuncs, iniGuess, tol):

Khởi tạo lagrangeMultipliers và lagrangeMultipliersH với giá trị ban đầu 1.0

penaltyParameter = 1.0

gamma = 1.5

penaltyMax = 10.0

x0 = iniGuess

doLoop = true

Trong khi doLoop:

Định nghĩa hàm Afunc(x1, x2):

Tính penalty ban đầu = Ffunc(x1, x2)

Cho mỗi ràng buộc gFuncs:

Cập nhật penalty bằng cách thêm các số nhân Lagrange và hình phạt

Cho mỗi ràng buộc HFuncs:

Cập nhật penalty bằng cách thêm các số nhân Lagrange và hình phạt bình phương

Tìm minX bằng POWELL\_METHOD(Afunc, x0, tol)

Nếu minX rỗng, trả về rỗng

In thông tin trạng thái (x0, số nhân Lagrange, penaltyParameter)

Nếu khoảng cách giữa minX và x0 < tol:

doLoop = false

minFuncValue = Afunc(minX[0], minX[1])

Ngược lại:

Cập nhật x0 = minX

Cập nhật lagrangeMultipliers và lagrangeMultipliersH

Tăng penaltyParameter bằng gamma, nhưng không vượt quá penaltyMax

Trả về {minX, minFuncValue}

Hàm MAIN:

Định nghĩa hàm mục tiêu Ffunc(x1, x2)

Định nghĩa danh sách các ràng buộc gFuncs

Định nghĩa danh sách các ràng buộc HFuncs (nếu có)

Xác định điểm bắt đầu iniGuess

Gọi AUGMENTED\_LAGRANGIAN để tìm nghiệm

In nghiệm và giá trị hàm mục tiêu