**KAUNAS UNIVERSITY OF TECHNOLOGY**

**FACULTY OF MATHEMATICS AND NATURAL SCIENCES**

**Module P160B116 “Optimization methods”**

Laboratory work #2 report

**Lecturer**

Mindaugas Šnipas

**Student**

Mykolas Paulauskas

**KAUNAS, 2021**

Contents

[1. Task 1 3](#_Toc69423384)

[1.1. Results 4](#_Toc69423385)

[1.2. Matlab code 4](#_Toc69423386)

[2. Task 2 7](#_Toc69423387)

[2.1. Results 7](#_Toc69423388)

[2.2. Matlab code: 8](#_Toc69423389)

[3. Task 3 10](#_Toc69423390)

[3.1. Results 10](#_Toc69423391)

[3.2. Matlab code 11](#_Toc69423392)

[4. Control work 13](#_Toc69423393)

Student number for the first laboratory work: 14

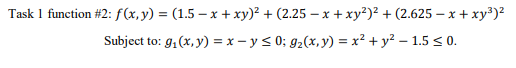


Figure 1 Function for part 1



Figure 2 Function for part 2

# Task 1

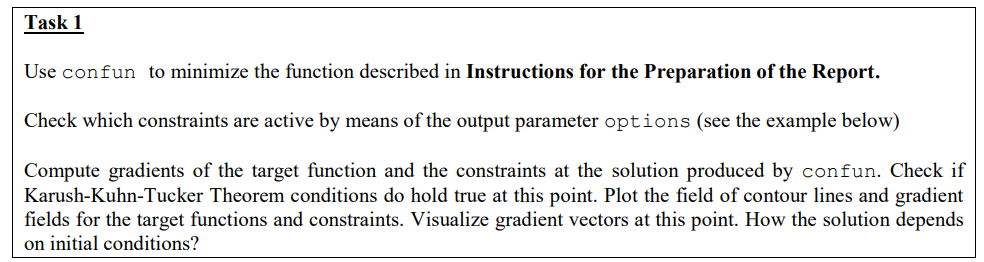


Figure 3 Task 1

## Results

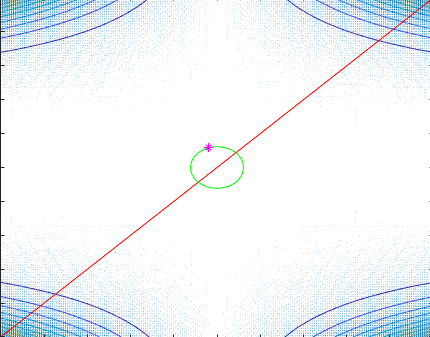


Figure 4 Results for task 1

From the figure shown above we can see both constrains graphed out the red one is and the green graph is . Local minimum value, which satisfies the constraints, of the target function was computed to be (-0.4036; 1.1563). The constraint at this value is passive and the constraint is active ().

KKT conditions apply, because with the computed gradients at the local minimum the lambda values are not all zeros and all are positive.

With different initial conditions the solution remains similar.

## Matlab code

From file **objfun.m**:

function f = objfun(x)

f = (1.5 - x(1) + x(1) \* x(2))^2 + (2.25 - x(1) + x(1) \* x(2)^2)^2 + (2.625 - x(1) + x(1) \* x(2)^3)^2;

From file **confun.m**:

function [c, ceq] = confun(x)

c = [x(1) - x(2); x(1)^2 + x(2)^2 - 1.5];

% since equality constraints are missing:

ceq = [];

From file **main.m**:

% use meshgrid to create a rectangular grid

[x, y] = meshgrid(-10:.1:10, -10:.1:10);

% compute function values at the points of the grid

z = (1.5 - x + x.\* y).^2 + (2.25 - x + x.\* y.^2).^2 + (2.625 - x + x.\* y.^3).^2;

% compute gradients

% dx – partial derivative in respect of x; dy – partial derivative in respect of y

[dx, dy] = gradient(z);

% plot contourlines

contour(x, y, z)

% the next plot will be constructed on top of the existing figure

hold on

% plot gradients

quiver(x, y, dx, dy)

% the next plot will be constructed on top of the existing figure

hold on

% plot the constraints function g1

x1 = -10:.1:10;

y1 = x1;

plot(x1, y1, 'r')

% the next plot will be constructed on top of the existing figure

hold on

% plot the constraints functions g2 positive values

x2 = -sqrt(1.5):.00001:sqrt(1.5);

y2 = sqrt(1.5 - x2.^2);

plot(x2, y2, 'g')

% the next plot will be constructed on top of the existing figure

hold on

% plot the constraints functions g2 negative values

plot(x2, -y2, 'g')

% make sure that the grid will stay 10x10

xlim([-10 10])

ylim([-10 10])

% make an initial guess:

x0 = [-1 1];

% Setup the optimization parameters:

% turn off large-scale algorithms

% turn on Display options for visualization of transient results

options = optimset('LargeScale', 'off', 'Display', 'iter');

% non-explicit constraints are replaced by []

[x, fval, exitflag, output] = fmincon('objfun', x0, [], [], [], [], [], [], 'confun', options);

% the next plot will be constructed on top of the existing figure

hold on

% plot the result X point

plot(x(1), x(2), 'm\*')

% finish drawing

hold off

disp(x)

disp(fval)

disp(exitflag)

disp(output.message)

# Task 2

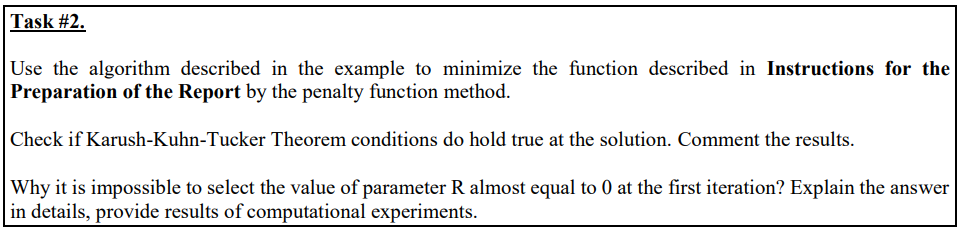


Figure 5 task 2

## Results

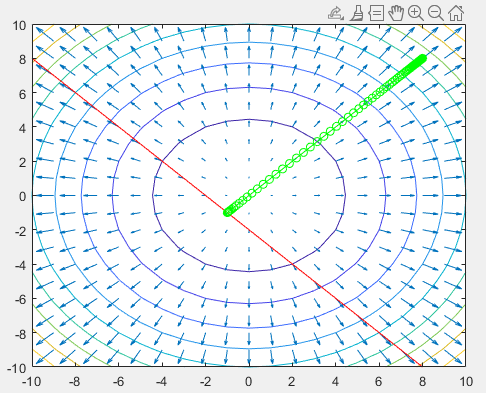


Figure 6 Results task 2

From the figure above we can see that see that the constrained minimum is reached with the value of approximately (-1, -1). At this value the constraint is active (equal to zero) there fore the KKT conditions apply.

The parameter R cannot be equal to almost zero because by dividing to a value close to zero we receive a value that is very big hence our penalty function step becomes too big to compute any reliable results.

## Matlab code:

% use meshgrid to create a rectangular grid

[x, y] = meshgrid(-10:1:10, -10:1:10);

% compute function values at the points of the grid

z = x.^2 + y.^2;

% compute gradients

% dx – partial derivative in respect of x; dy – partial derivative in respect of y

[dx, dy] = gradient(z);

% plot contourlines

contour(x, y, z)

% the next plot will be constructed on top of the existing figure

hold on

% plot gradients

quiver(x, y, dx, dy)

% the next plot will be constructed on top of the existing figure

hold on

% plot the constraints function g1

x1 = -10:1:10;

y1 = -2 - x1;

plot(x1, y1, 'r')

% the next plot will be constructed on top of the existing figure

hold on

% make sure that the grid will stay 10x10

xlim([-10 10])

ylim([-10 10])

% penalty parameter R

r = 1;

% precision

epsilon = 0.001;

% starting point

xs = [8 8];

% the step size

gamma = 0.0001;

% limit the number of iterations in order to avoid infinite loops

max\_iterations = 10000;

for i=1:max\_iterations

% gradient of the transformed function

e1 = 2\*xs(1) + 2 \* (xs(1) + xs(2) + 2) / r;

e2 = 2\*xs(2) + 2 \* (xs(1) + xs(2) + 2) / r;

grad = [e1 e2];

% compute the next point

x = xs - gamma\*grad;

% plot the step

plot([xs(1), x(1)], [xs(2), x(2)], 'g', [xs(1), x(1)], [xs(2), x(2)], 'go')

% distance between the previous and current point

distance = sqrt((x(1) - xs(1))^2 + (x(2) - xs(2))^2);

% refresh the variable

xs = x;

% update the parameter R

r = r/5;

% Check if the current minimum point did move from the previous minimum

% point less than epsilon

if distance < epsilon

break;

end

end

# Task 3

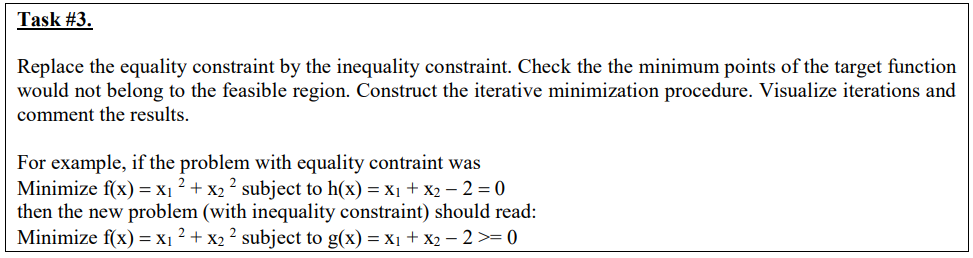


Figure 7 Task 3

## Results

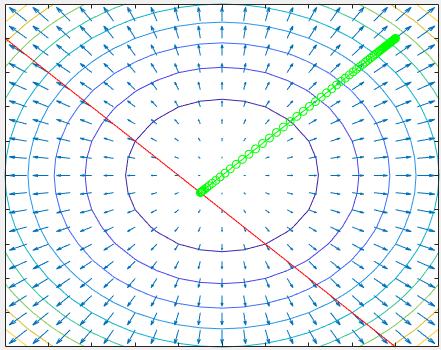


Figure 8 Results of task 3

When replacing the equality constraint by the inequality constraint I needed to make the function into because minimumo of the target function is (0, 0). So the result doesn‘t differ a lot from the task 2.

## Matlab code

% use meshgrid to create a rectangular grid

[x, y] = meshgrid(-10:1:10, -10:1:10);

% compute function values at the points of the grid

z = x.^2 + y.^2;

% compute gradients

% dx – partial derivative in respect of x; dy – partial derivative in respect of y

[dx, dy] = gradient(z);

% plot contourlines

contour(x, y, z)

% the next plot will be constructed on top of the existing figure

hold on

% plot gradients

quiver(x, y, dx, dy)

% the next plot will be constructed on top of the existing figure

hold on

% plot the constraints function g1

x1 = -10:1:10;

y1 = -2 - x1;

plot(x1, y1, 'r')

% the next plot will be constructed on top of the existing figure

hold on

% make sure that the grid will stay 10x10

xlim([-10 10])

ylim([-10 10])

% penalty parameter R

r = 1;

% precision

epsilon = 0.001;

% starting point

xs = [8 8];

% the step size

gamma = 0.0001;

% limit the number of iterations in order to avoid infinite loops

max\_iterations = 10000;

for i=1:max\_iterations

% penalty

p = max(0, xs(1)+xs(2)+2);

% gradient of the transformed function

grad = [2\*xs(1) 2\*xs(2)] + 2/r\*p\*[1 1];

% compute the next point

x = xs - gamma\*grad;

% plot the step

plot([xs(1), x(1)], [xs(2), x(2)], 'g', [xs(1), x(1)], [xs(2), x(2)], 'go')

% distance between two points

distance = sqrt((x(1) - xs(1))^2 + (x(2) - xs(2))^2);

% refresh the variable

xs = x;

% update the parameter R

r = r/1.1;

% Check if the current minimum point did move from the previous minimum

% point less than epsilon

if distance < epsilon

break;

end

end

# Control work