**KAUNAS UNIVERSITY OF TECHNOLOGY**

**FACULTY OF MATHEMATICS AND NATURAL SCIENCES**

**Module P160B116 “Optimization methods”**

Laboratory work #1 report

**Lecturer**

Mindaugas Šnipas

**Student**

Mykolas Paulauskas

**KAUNAS, 2021**

Contents

[1. Task 1 3](#_Toc67057776)

[1.1. Results 4](#_Toc67057777)

[1.2. Matlab code 4](#_Toc67057778)

[2. Task 2 4](#_Toc67057779)

[2.1. Results 5](#_Toc67057780)

[2.2. Matlab code: 7](#_Toc67057781)

[3. Task 3 8](#_Toc67057782)

[3.1. Results 9](#_Toc67057783)

[3.2. Matlab code 9](#_Toc67057784)

[4. Task 4 10](#_Toc67057785)

[4.1. Results 10](#_Toc67057786)

[4.2. Matlab code 11](#_Toc67057787)

[5. Task 5 12](#_Toc67057788)

[5.1. Results 12](#_Toc67057789)

[5.2. Matlab code 12](#_Toc67057790)

[6. Task 6 part 1 13](#_Toc67057791)

[6.1. Results 14](#_Toc67057792)

[6.2. Matlab code 14](#_Toc67057793)

[7. Task 6 part two 17](#_Toc67057794)

[7.1. Results 18](#_Toc67057795)

[7.2. Matlab code 18](#_Toc67057796)

[8. Task 6 part three 21](#_Toc67057797)

[8.1. Results 21](#_Toc67057798)

[8.2. Matlab code 21](#_Toc67057799)

[9. Control work 24](#_Toc67057800)

[9.1. Task 1 24](#_Toc67057801)

[9.2. Task 2 25](#_Toc67057802)

[9.3. Task 3 26](#_Toc67057803)

[9.4. Task 4 27](#_Toc67057804)

Student number for the first laboratory work: 14

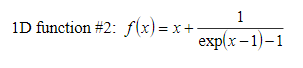


Figure 1 Function for part 1



Figure 2 Function for part 2

# Task 1

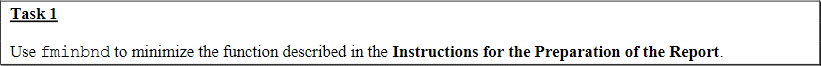


Figure 3 Task 1

## Results

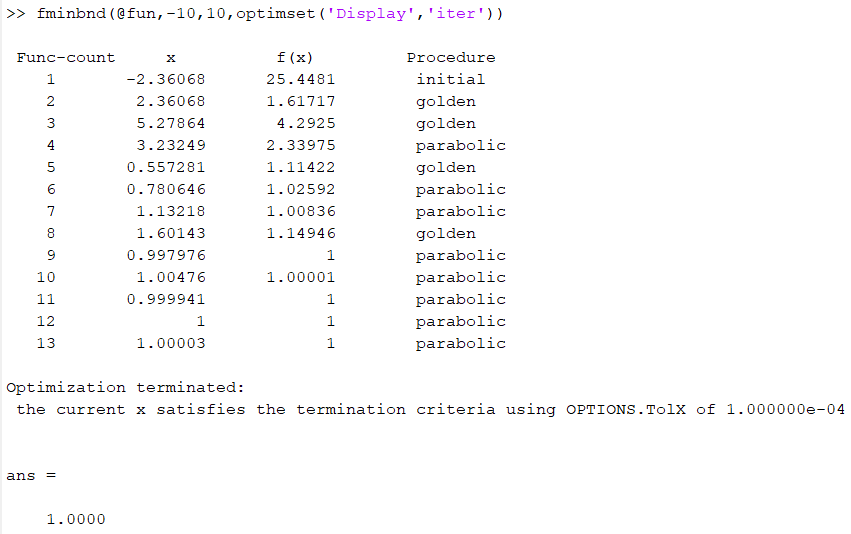


Figure 4 Results for task 1

## Matlab code

function f = fun(x)

f = x + (1/exp(x - 1) - 1);

# Task 2

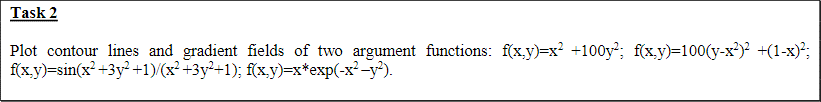


Figure 5 task 2

## Results

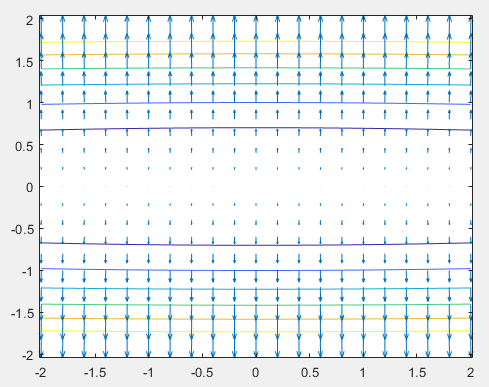


Figure 6 Results of first function

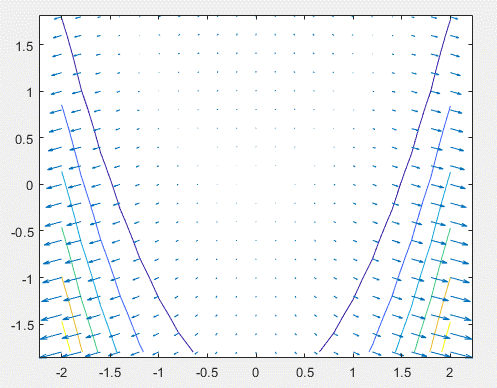


Figure 7 Results of second function

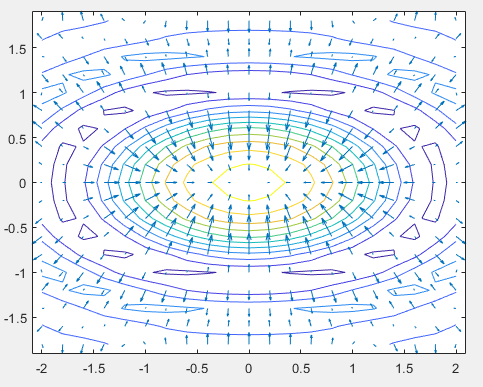


Figure 8 Results of the third function

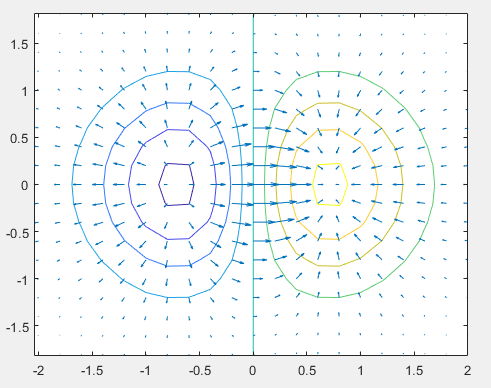


Figure 9 Results of the last function

## Matlab code:

% use meshgrid to create a rectangular grid

[x,y]=meshgrid(-2:.2:2,-1.8:.2:1.8);

% compute function values at the points of the grid

%z = x.^2 + 100 \* y.^2;

%z = 100 \* (y - x.^2).^2 + (1-x).^2;

%z = sin(x.^2 + 3\*y.^2 + 1) ./ (x.^2 + 3\*y.^2 + 1);

z=x.\*exp(-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)

% finish drawing

hold off

# Task 3

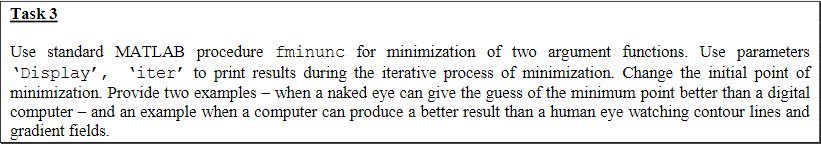


Figure 10 Task 3

## Results

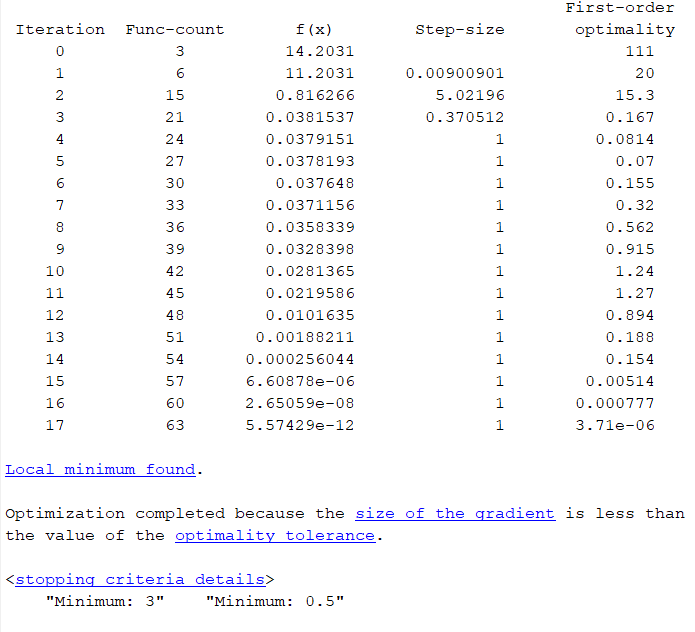


Figure 11 Results of task 3

## Matlab code

From file fun.m:

function f = fun(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 main.m:

x0 = [4,1]; % guess the initial point

options = optimset('LargeScale','off', 'Display', 'iter'); % no LargeScale functions

[x,fval,exitflag,output] = fminunc(@fun,x0,options);

disp("Minimum: " + x);

# Task 4

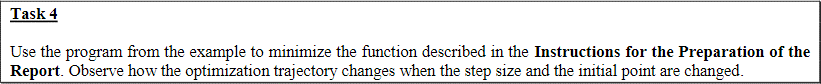


Figure 12 Task 4

## Results

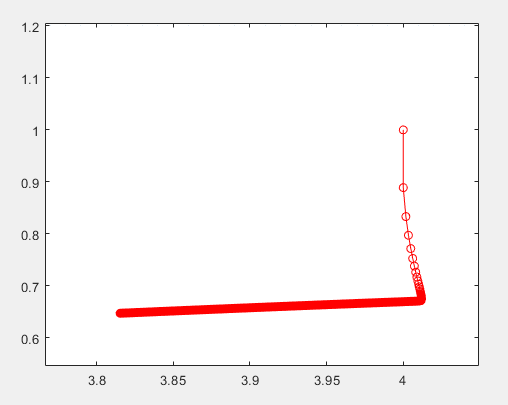


Figure 13 Task 4 results with starting value [4 , 1]

Computed results: minimum is at [3.8154, 0.6473]

## Matlab code

From gradient1.m file:

function g = gradient1(x, y)

df\_dx = 2\*(y^2 - 1)\*(x\*y^2 - x + 9/4) + 2\*(y^3 - 1)\*(x\*y^3 - x + 21/8) + 2\*(y - 1)\*(x\*y - x + 3/2);

df\_dy = 2\*x\*(x\*y - x + 3/2) + 4\*x\*y\*(x\*y^2 - x + 9/4) + 6\*x\*y^2\*(x\*y^3 - x + 21/8);

g = [df\_dx, df\_dy];

From fun.m file:

% Plot the trajectory of Anti-Gradient Descent

[x,y]=meshgrid(-3:.01:5);

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

figure(1)

hold off

contour(x,y,z)

[dx,dy]=gradient(z,.2,.2);

hold on

quiver(x,y,dx,dy)

% set the initial point

x=[4,1];

xs=x; % dummy variable required for the iterative process

itsk=2500; % the number of iterations

step=0.001; % the step size

for i=1:itsk

% compute the next point

x = xs - step\*gradient1(xs(1), xs(2));

% plot the step

plot([xs(1),x(1)],[xs(2),x(2)],'r',[xs(1),x(1)],[xs(2),x(2)],'ro')

% refresh the variables

xs=x;

end

disp(xs);

# Task 5

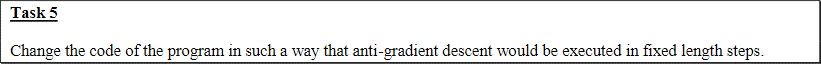


Figure 14 Task 5

## Results

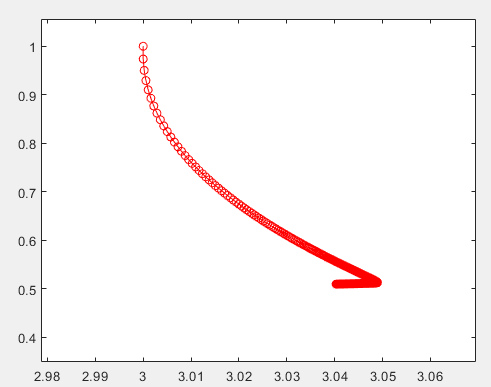


Figure 15 Results of task 5 with starting value [4, 1]

Computed results: minimum is at [3.0403, 0.5099]

## Matlab code

From gradient1.m file:

function g = gradient1(x, y)

df\_dx = 2\*(y^2 - 1)\*(x\*y^2 - x + 9/4) + 2\*(y^3 - 1)\*(x\*y^3 - x + 21/8) + 2\*(y - 1)\*(x\*y - x + 3/2);

df\_dy = 2\*x\*(x\*y - x + 3/2) + 4\*x\*y\*(x\*y^2 - x + 9/4) + 6\*x\*y^2\*(x\*y^3 - x + 21/8);

g = [df\_dx, df\_dy];

From gradientLength.m file:

function l = gradientLength(x, y)

l = sqrt(x^2+y^2);

From fun.m file:

% Plot the trajectory of Anti-Gradient Descent

[x,y]=meshgrid(-3:.01:5);

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

figure(1)

hold off

contour(x,y,z)

[dx,dy]=gradient(z,.2,.2);

hold on

quiver(x,y,dx,dy)

% set the initial point

x=[4,1];

xs=x; % dummy variable required for the iterative process

itsk=2500; % the number of iterations

step=0.001; % the step size

for i=1:itsk

% compute the next point

x = xs - step\*gradient1(xs(1), xs(2))/ /gradientLength(xs(1), xs(2));

% plot the step

plot([xs(1),x(1)],[xs(2),x(2)],'r',[xs(1),x(1)],[xs(2),x(2)],'ro')

% refresh the variables

xs=x;

end

disp(xs);

# Task 6 part 1

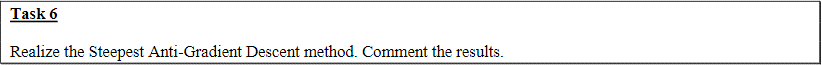


Figure task 6a

## Results

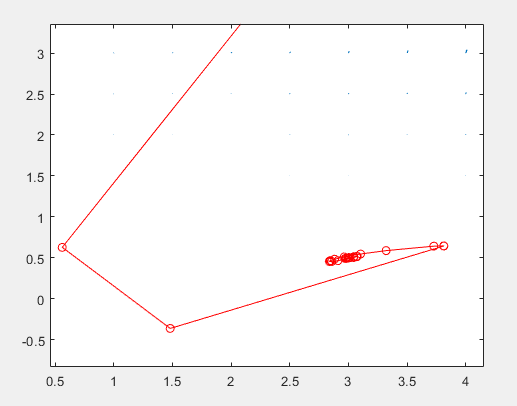


Figure Task 6a results with starting values [3, 5]

Computed value: minimum is at [3, 0.5]

From the graph we can see that the steepest descent method doesn’t change the gradient direction until it stops getting lower. This results in the graph making fewer, but sharper turns (when compared to AGD).

## Matlab code

From length1.m file:

function l=length1(x)

l = sqrt(x(1)^2 + x(2)^2);

From golden\_section\_search.m file:

function gamma = golden\_section\_search(xs, s)

a=0; % start of interval

b=1; % end of interval

epsilon=0.000001; % accuracy value

iter= 50; % maximum number of iterations

tau=double((sqrt(5)-1)/2); % golden proportion coefficient, around 0.618

k=0; % number of iterations

x1=a+(1-tau)\*(b-a); % computing x values

x2=a+tau\*(b-a);

f\_x1=f\_gamma(xs, s, x1);

f\_x2=f\_gamma(xs, s, x2);

while ((abs(b-a)>epsilon) && (k<iter))

k=k+1;

if(f\_x1<f\_x2)

b=x2;

x2=x1;

x1=a+(1-tau)\*(b-a);

f\_x1=f\_gamma(xs, s, x1);

f\_x2=f\_gamma(xs, s, x2);

else

a=x1;

x1=x2;

x2=a+tau\*(b-a);

f\_x1=f\_gamma(xs, s, x1);

f\_x2=f\_gamma(xs, s, x2);

end

k=k+1;

end

% chooses minimum point

if(f\_x1<f\_x2)

gamma = x1;

else

gamma = x2;

end

From f\_gamma.m file:

function z = f\_gamma(x0, s, gamma)

x = x0 + gamma\*s;

z = f(x(1), x(2));

From df.m file:

function g = df(x, y)

df\_dx = 2\*(y^2 - 1)\*(x\*y^2 - x + 9/4) + 2\*(y^3 - 1)\*(x\*y^3 - x + 21/8) + 2\*(y - 1)\*(x\*y - x + 3/2);

df\_dy = 2\*x\*(x\*y - x + 3/2) + 4\*x\*y\*(x\*y^2 - x + 9/4) + 6\*x\*y^2\*(x\*y^3 - x + 21/8);

g = [df\_dx, df\_dy];

From fun.m file:

% Plot the trajectory of Steepest Descent

[x, y] = meshgrid(-5:.5:5);

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

figure(1)

hold off

contour(x, y, z)

[dx, dy] = gradient(z,.2,.2);

hold on

quiver(x, y, dx, dy)

% set the initial point

x0 = [3, 5];

xs = x0; % dummy variable required for the iterative process

itsk = 50; % the number of iterations

s = -df(x0(1), x0(2));

for i=1:itsk

% calculate gamma/step

gamma = golden\_section\_search(xs, s);

% compute the next point

x = xs + gamma\*s;

% plot the step

plot([xs(1),x(1)],[xs(2),x(2)],'r',[xs(1),x(1)],[xs(2),x(2)],'ro')

g\_1 = df(x(1), x(2));

len\_1 = length1(g\_1)^2;

g\_2 = df(xs(1), xs(2));

len\_2 = length1(g\_2)^2;

b = len\_1/len\_2;

% calculate gradient

g = df(x(1), x(2));

s = -g + b\*s;

% refresh the variables

xs = x;

disp("Iteration: " + i + " x: (" + xs(1) + " ; " + xs(2) + ")")

end

# Task 6 part two

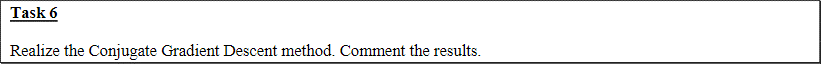


Figure Task 6b

## Results

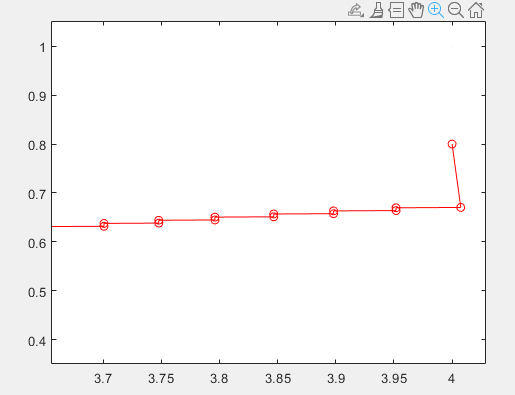


Figure result of task 6b

Computed results: minimum is at [3, 0.5]

When compared to the steepest descent method we see that the steps taken by the algorithm don’t backtrack to previous values e.g. it does not move in two horizontal and vertical directions and only goes in one horizontal and one vertical direction.

## Matlab code

From golden\_section\_search.m file:

function gamma = golden\_section\_search(xs)

a=0; % start of interval

b=1; % end of interval

epsilon=0.000001; % accuracy value

iter= 50; % maximum number of iterations

tau=double((sqrt(5)-1)/2); % golden proportion coefficient, around 0.618

k=0; % number of iterations

x1=a+(1-tau)\*(b-a); % computing x values

x2=a+tau\*(b-a);

f\_x1=f\_gamma(xs, x1);

f\_x2=f\_gamma(xs, x2);

while ((abs(b-a)>epsilon) && (k<iter))

k=k+1;

if(f\_x1<f\_x2)

b=x2;

x2=x1;

x1=a+(1-tau)\*(b-a);

f\_x1=f\_gamma(xs, x1);

f\_x2=f\_gamma(xs, x2);

else

a=x1;

x1=x2;

x2=a+tau\*(b-a);

f\_x1=f\_gamma(xs, x1);

f\_x2=f\_gamma(xs, x2);

end

k=k+1;

end

% chooses minimum point

if(f\_x1<f\_x2)

gamma = x1;

else

gamma = x2;

end

From f\_gamma.m file:

function z = f\_gamma(x0, gamma)

x = x0 - gamma\*df(x0(1), x0(2));

z = f(x(1), x(2));

From df.m file:

function g = df(x, y)

df\_dx = 2\*(y^2 - 1)\*(x\*y^2 - x + 9/4) + 2\*(y^3 - 1)\*(x\*y^3 - x + 21/8) + 2\*(y - 1)\*(x\*y - x + 3/2);

df\_dy = 2\*x\*(x\*y - x + 3/2) + 4\*x\*y\*(x\*y^2 - x + 9/4) + 6\*x\*y^2\*(x\*y^3 - x + 21/8);

g = [df\_dx, df\_dy];

From fun.m file:

% Plot the trajectory of conjugate Gradient Descent

[x, y] = meshgrid(-5:.5:5);

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

figure(1)

hold off

contour(x, y, z)

[dx, dy] = gradient(z,.2,.2);

hold on

quiver(x, y, dx, dy)

% set the initial point

x0=[4, 0.8];

xs=x0; % dummy variable required for the iterative process

itsk=50; % the number of iterations

for i=1:itsk

% calculate gradient

g = df(xs(1), xs(2));

% calculate gamma/step

gamma = golden\_section\_search(xs);

% compute the next point

x = xs - gamma\*g;

% plot the step

plot([xs(1),x(1)],[xs(2),x(2)],'r',[xs(1),x(1)],[xs(2),x(2)],'ro')

% refresh the variables

xs=x;

end

# Task 6 part three



Figure task 6c

## Results

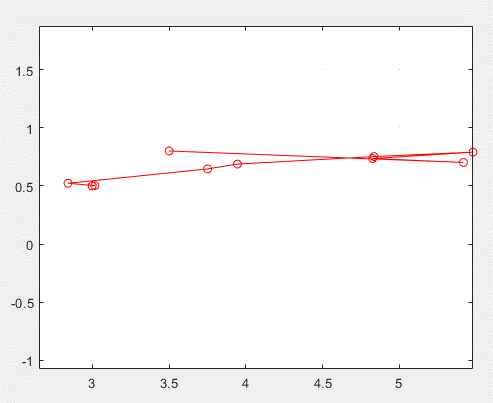


Figure task 6c results

Computed results: minimum is at [3, 0.5]

Similar to CG method, the Newtons method has a “zig-zag” pattern, although the directions and angles of each step differ drastically.

## Matlab code

From file hessian.m :

function h = hessian(x, y)

xx = (y^2 - 1)\*(2\*y^2 - 2) + (y^3 - 1)\*(2\*y^3 - 2) + (2\*y - 2)\*(y - 1);

xy = x\*(2\*y - 2) - 2\*x + 6\*y^2\*(x\*y^3 - x + 21/8) + 2\*x\*y + 4\*y\*(x\*y^2 - x + 9/4) + 3\*x\*y^2\*(2\*y^3 - 2) + 2\*x\*y\*(2\*y^2 - 2) + 3;

yy = 8\*x^2\*y^2 + 18\*x^2\*y^4 + 4\*x\*(x\*y^2 - x + 9/4) + 2\*x^2 + 12\*x\*y\*(x\*y^3 - x + 21/8);

yx = 6\*y^2\*(x\*y^3 - x + 21/8) - 2\*x + 2\*x\*y + 2\*x\*(y - 1) + 4\*y\*(x\*y^2 - x + 9/4) + 4\*x\*y\*(y^2 - 1) + 6\*x\*y^2\*(y^3 - 1) + 3;

h = [xx xy;

yx yy];

From gradient1.m file:

function g = gradient1(x, y)

df\_dx = 2\*(y^2 - 1)\*(x\*y^2 - x + 9/4) + 2\*(y^3 - 1)\*(x\*y^3 - x + 21/8) + 2\*(y - 1)\*(x\*y - x + 3/2);

df\_dy = 2\*x\*(x\*y - x + 3/2) + 4\*x\*y\*(x\*y^2 - x + 9/4) + 6\*x\*y^2\*(x\*y^3 - x + 21/8);

g = [df\_dx, df\_dy];

From fun.m file:

% Plot the trajectory of Newtons method

[x,y]=meshgrid(-5:.5:5);

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

figure(1)

hold off

contour(x,y,z)

[dx,dy]=gradient(z,.2,.2);

hold on

quiver(x,y,dx,dy)

% set the initial point

x=[3.5,0.8];

xs=x; % dummy variable required for the iterative process

itsk=20; % the number of iterations

for i=1:itsk

h = hessian(xs(1), xs(2));

disp(h)

% compute the next point

x = xs - gradient1(xs(1), xs(2)) \* inv(h);

% plot the step

plot([xs(1),x(1)],[xs(2),x(2)],'r',[xs(1),x(1)],[xs(2),x(2)],'ro')

% refresh the variables

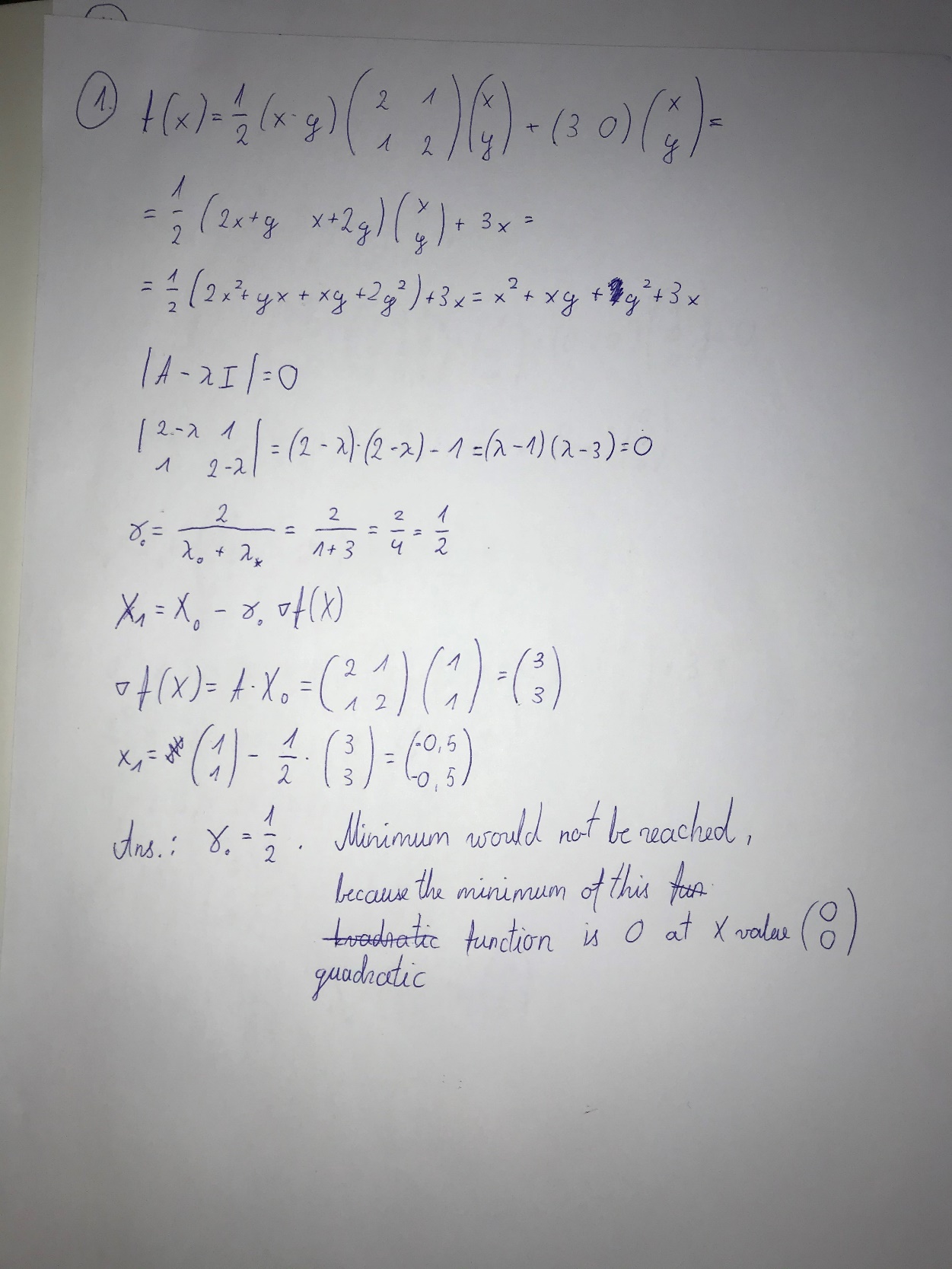
xs=x;

disp("Iteration: " + i + " Minimum: (" + xs(1) + " ; " + xs(2) + ")")

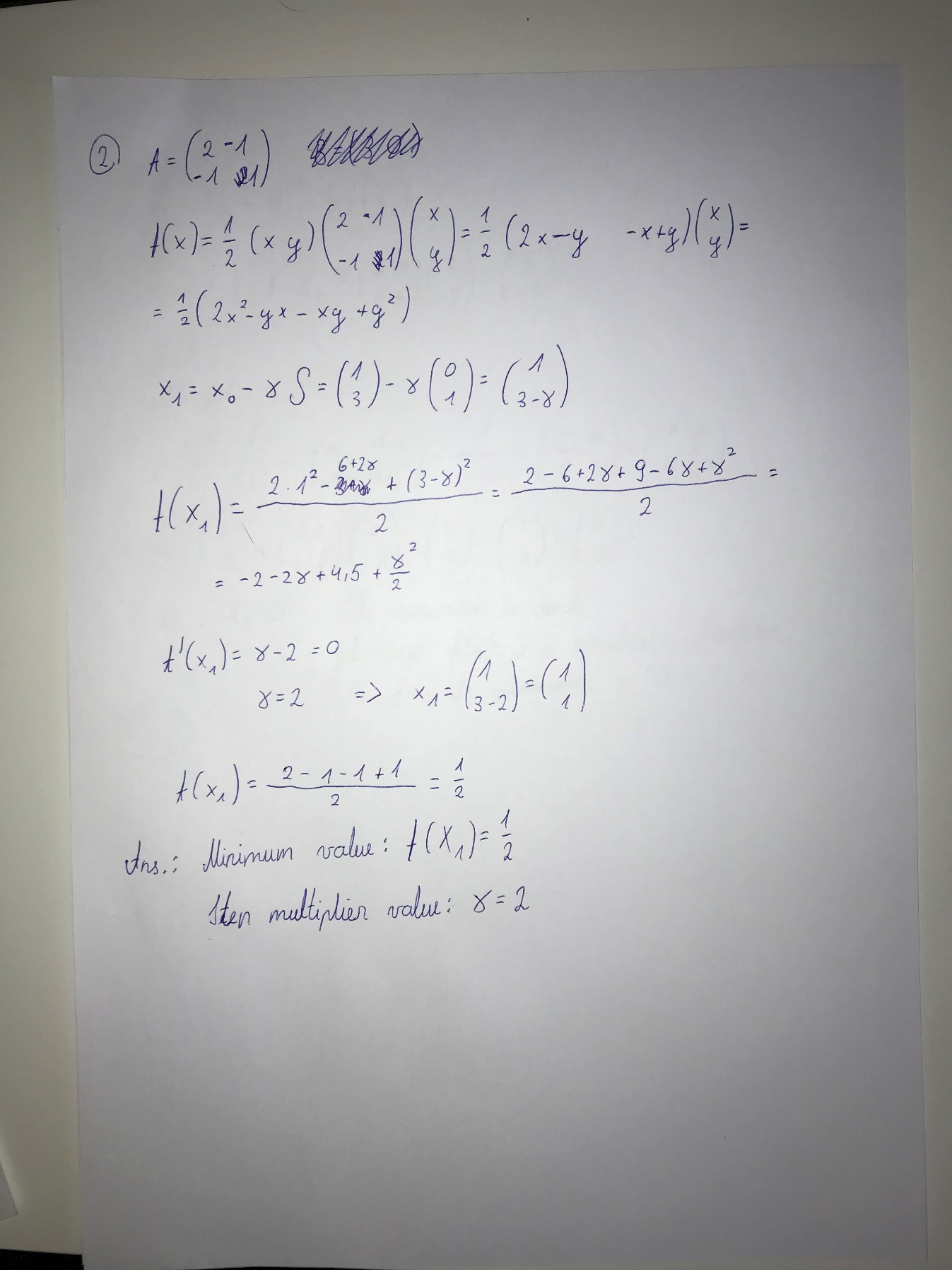
end

# Control work

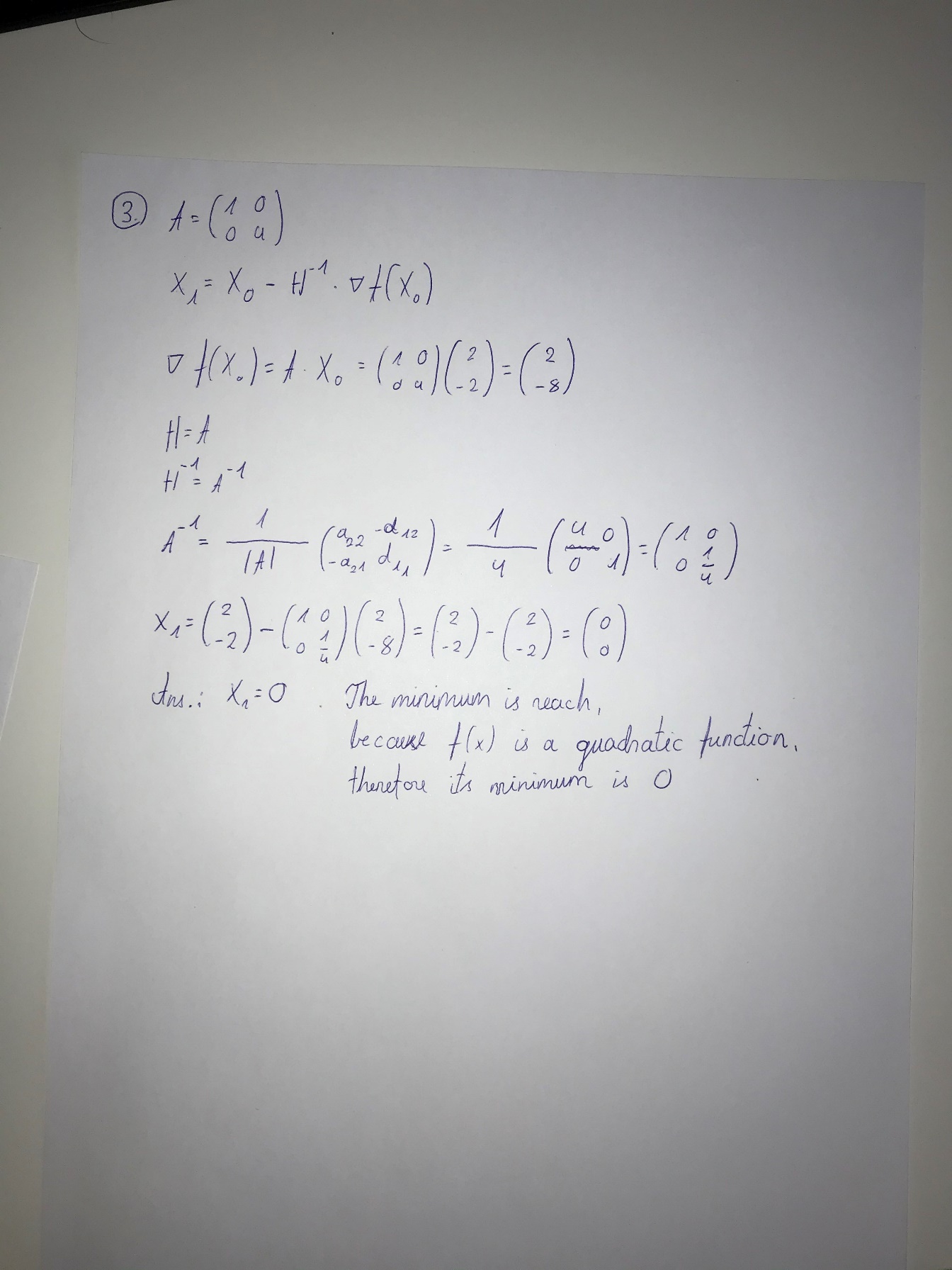
## Task 1



## Task 2



## Task 3



## Task 4

