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

Student number for the first laboratory work: 14

$$\text{1D function \#2: } f(x) = x + \frac{1}{\exp(x-1)-1}$$

Figure 1 Function for part 1

$$\text{2D function \#4: Beale's function: } f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$

Figure 2 Function for part 2

1. Task 1

<u>Task 1</u>
Use `fminbnd` to minimize the function described in the **Instructions for the Preparation of the Report**.

Figure 3 Task 1

1.1. Results

```
>> fminbnd(@fun,-10,10,optimset('Display','iter'))

 Func-count     x          f(x)         Procedure
    1        -2.36068     25.4481        initial
    2         2.36068     1.61717        golden
    3         5.27864      4.2925        golden
    4         3.23249     2.33975        parabolic
    5        0.557281     1.11422        golden
    6        0.780646     1.02592        parabolic
    7         1.13218     1.00836        parabolic
    8         1.60143     1.14946        golden
    9        0.997976           1        parabolic
   10         1.00476     1.00001        parabolic
   11        0.999941           1        parabolic
   12               1           1        parabolic
   13         1.00003           1        parabolic

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-04


ans =

    1.0000
```

Figure 4 Results for task 1

1.2. Matlab code

```
function f = fun(x)
f = x + (1/exp(x - 1) - 1);
```

2. Task 2

Task 2

Plot contour lines and gradient fields of two argument functions: $f(x,y)=x^2+100y^2$; $f(x,y)=100(y-x^2)^2+(1-x)^2$; $f(x,y)=\sin(x^2+3y^2+1)/(x^2+3y^2+1)$; $f(x,y)=x*\exp(-x^2-y^2)$.

Figure 5 task 2

2.1. Results

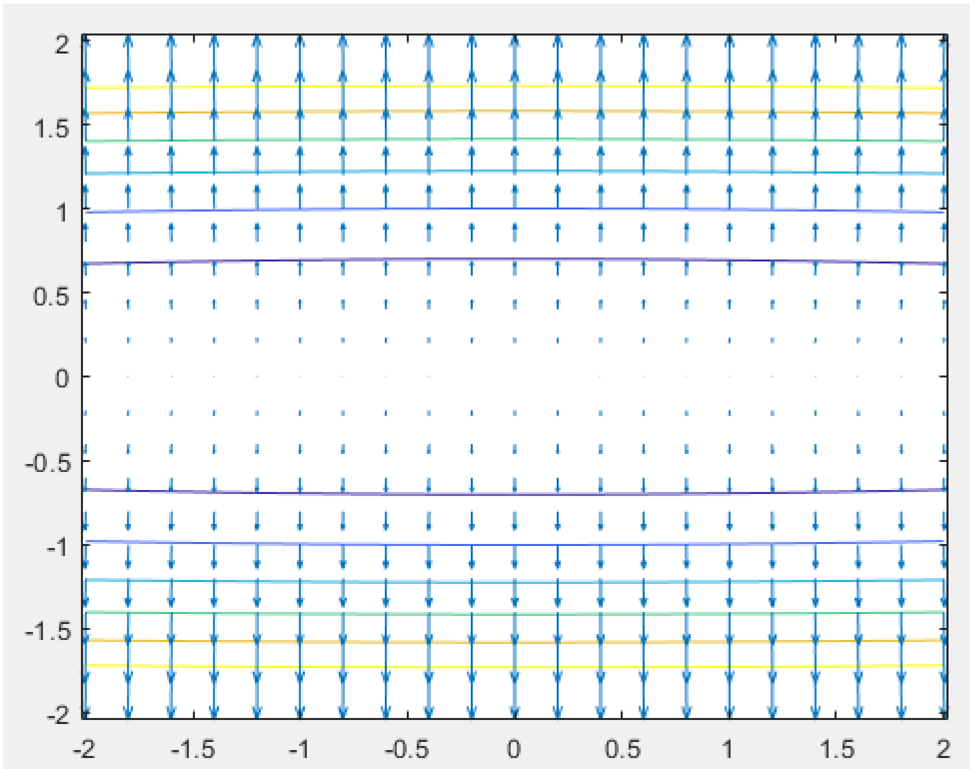

Figure 6 Results of first function

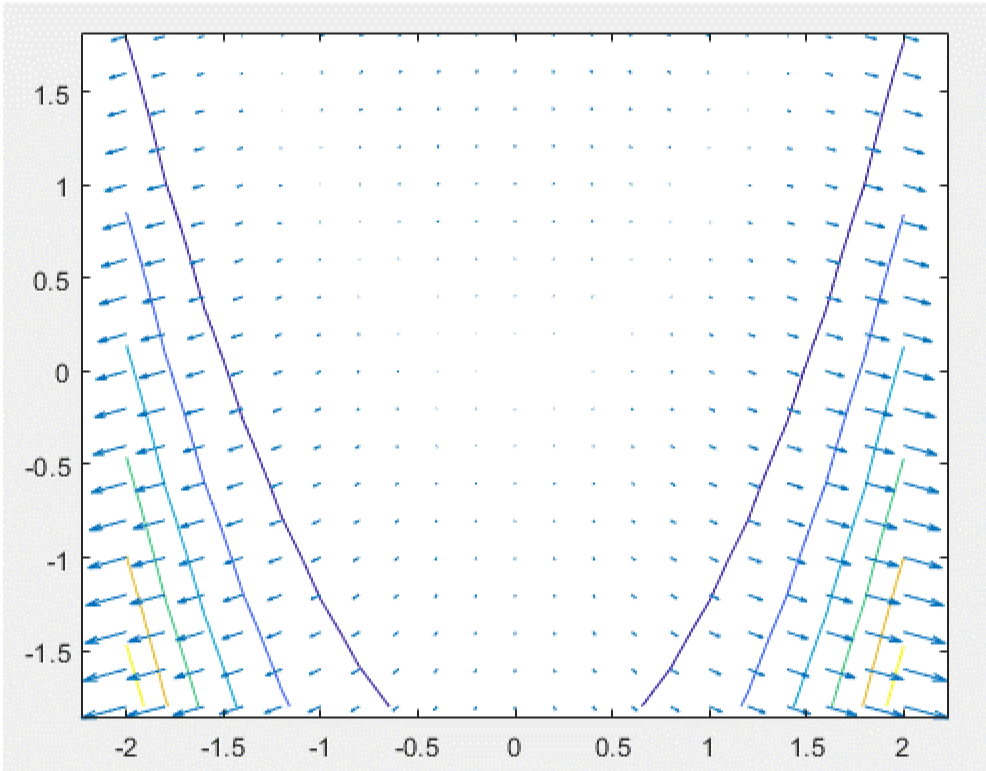

Figure 7 Results of second function

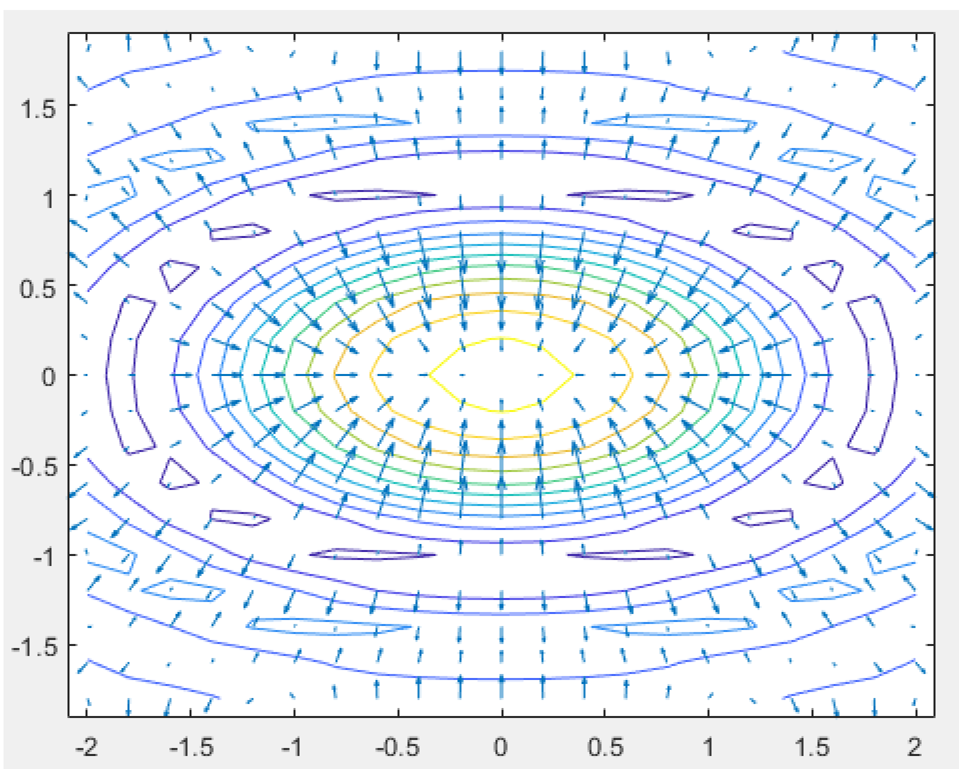

Figure 8 Results of the third function

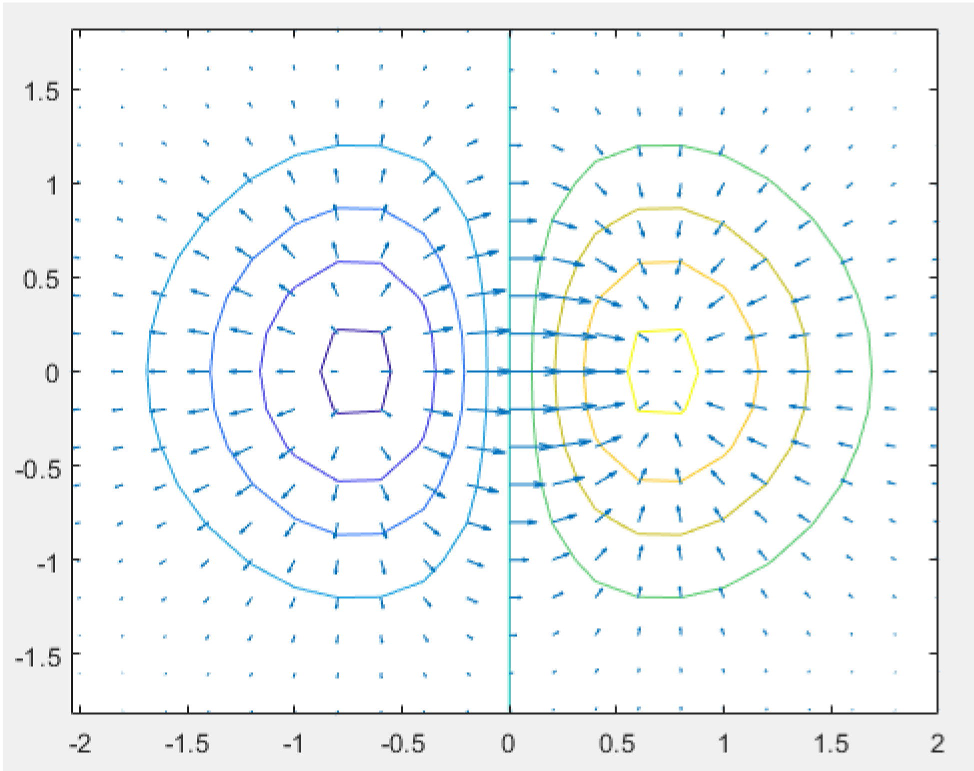

Figure 9 Results of the last function

2.2. 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
```

3. Task 3

Task 3

Use standard MATLAB procedure `fminunc` for minimization of two argument functions. Use parameters `'Display'`, `'iter'` to print results during the iterative process of minimization. Change the initial point of minimization. Provide two examples – when a naked eye can give the guess of the minimum point better than a digital computer – and an example when a computer can produce a better result than a human eye watching contour lines and gradient fields.

Figure 10 Task 3

3.1. Results

Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	14.2031		111
1	6	11.2031	0.00900901	20
2	15	0.816266	5.02196	15.3
3	21	0.0381537	0.370512	0.167
4	24	0.0379151	1	0.0814
5	27	0.0378193	1	0.07
6	30	0.037648	1	0.155
7	33	0.0371156	1	0.32
8	36	0.0358339	1	0.562
9	39	0.0328398	1	0.915
10	42	0.0281365	1	1.24
11	45	0.0219586	1	1.27
12	48	0.0101635	1	0.894
13	51	0.00188211	1	0.188
14	54	0.000256044	1	0.154
15	57	6.60878e-06	1	0.00514
16	60	2.65059e-08	1	0.000777
17	63	5.57429e-12	1	3.71e-06

Local minimum found.

Optimization completed because the size of the gradient is less than the value of the optimality tolerance.

<stopping criteria details>

"Minimum: 3"   "Minimum: 0.5"

Figure 11 Results of task 3

3.2. 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);
```

4. Task 4

Task 4

Use the program from the example to minimize the function described in the **Instructions for the Preparation of the Report**. Observe how the optimization trajectory changes when the step size and the initial point are changed.

Figure 12 Task 4

4.1. Results

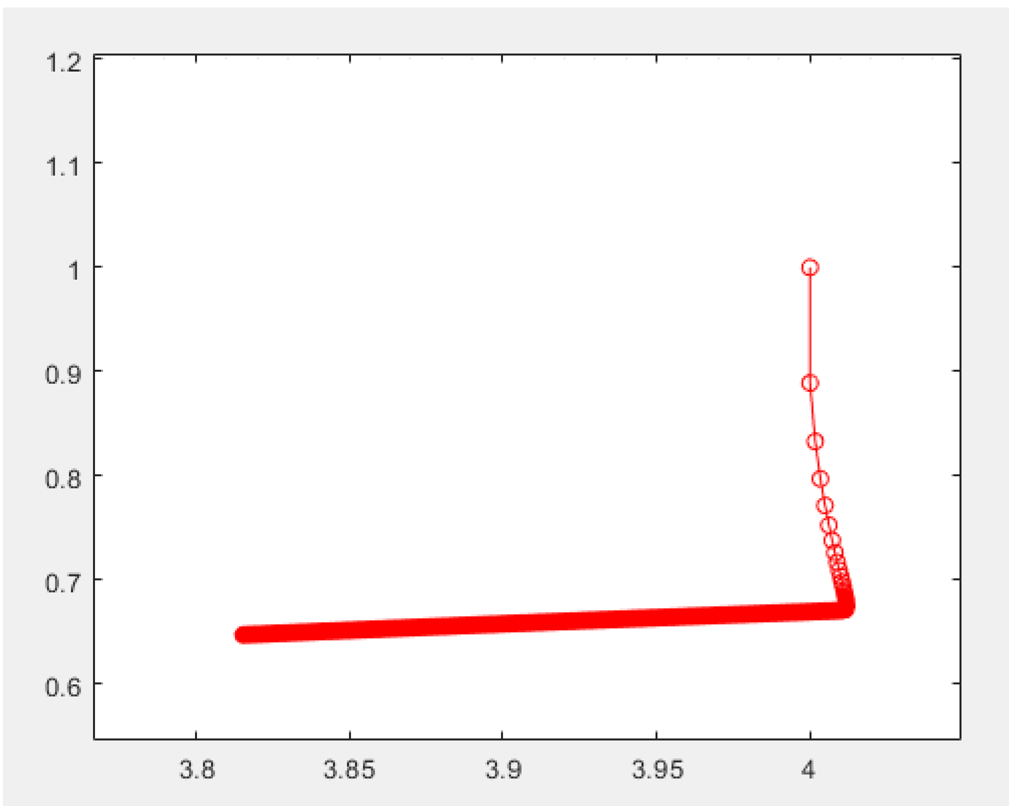

Figure 13 Task 4 results with starting value [4 , 1]

Computed results: minimum is at [3.8154, 0.6473]

4.2. 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);
```

5. Task 5

Task 5

Change the code of the program in such a way that anti-gradient descent would be executed in fixed length steps.

Figure 14 Task 5

5.1. Results

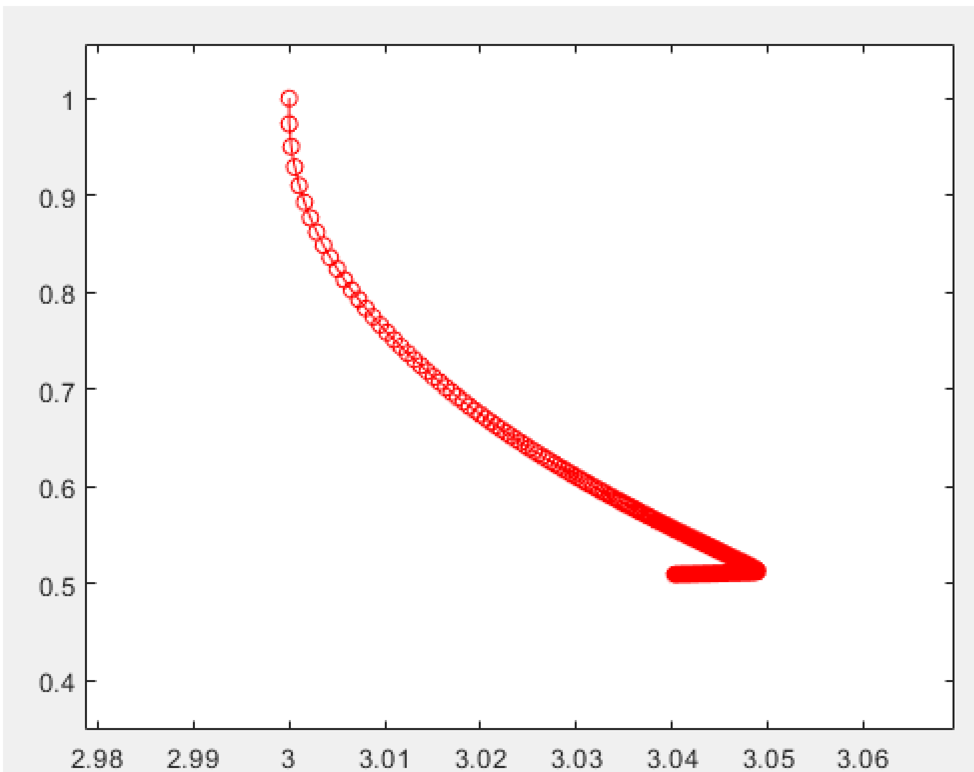

Figure 15 Results of task 5 with starting value [4, 1]

Computed results: minimum is at [3.0403, 0.5099]

5.2. 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);
```

6. Task 6 part 1

Task 6

Realize the Steepest Anti-Gradient Descent method. Comment the results.

Figure 16 task 6a

```
a=0;
```

6.1. Results

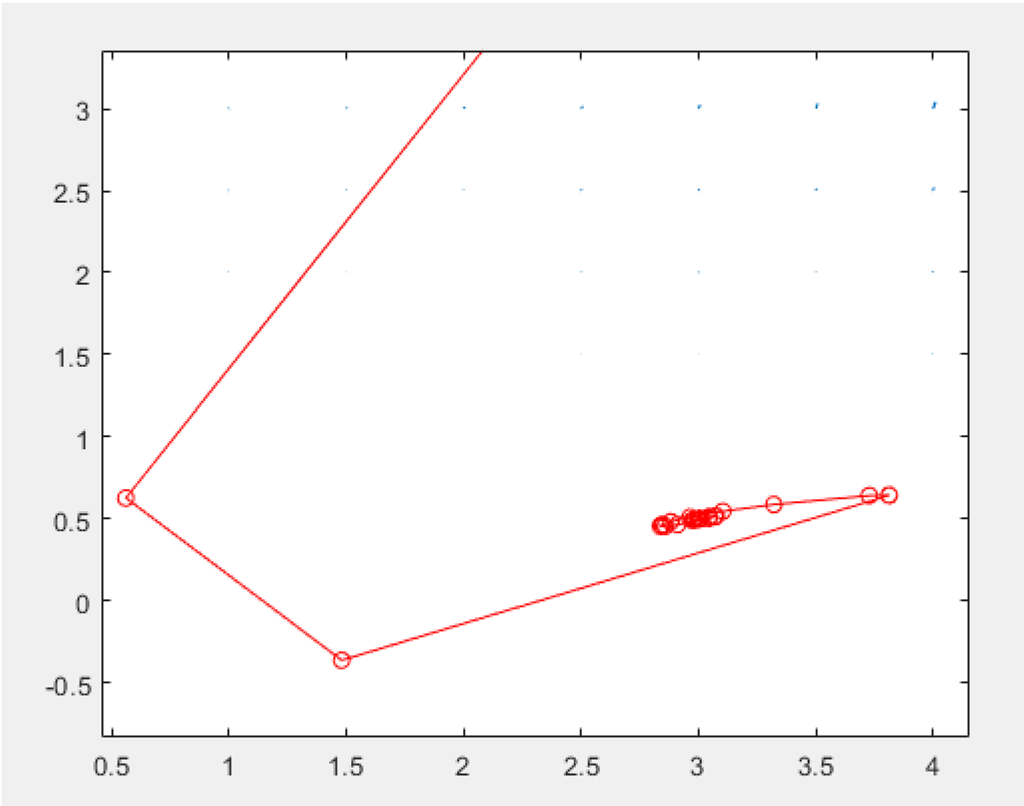

Figure 17 Task 6a results with starting values [3, 5]

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).

6.2. 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
```

7. Task 6 part two

Task 6
Realize the Conjugate Gradient Descent method. Comment the results.

Figure 18 Task 6b

7.1. Results

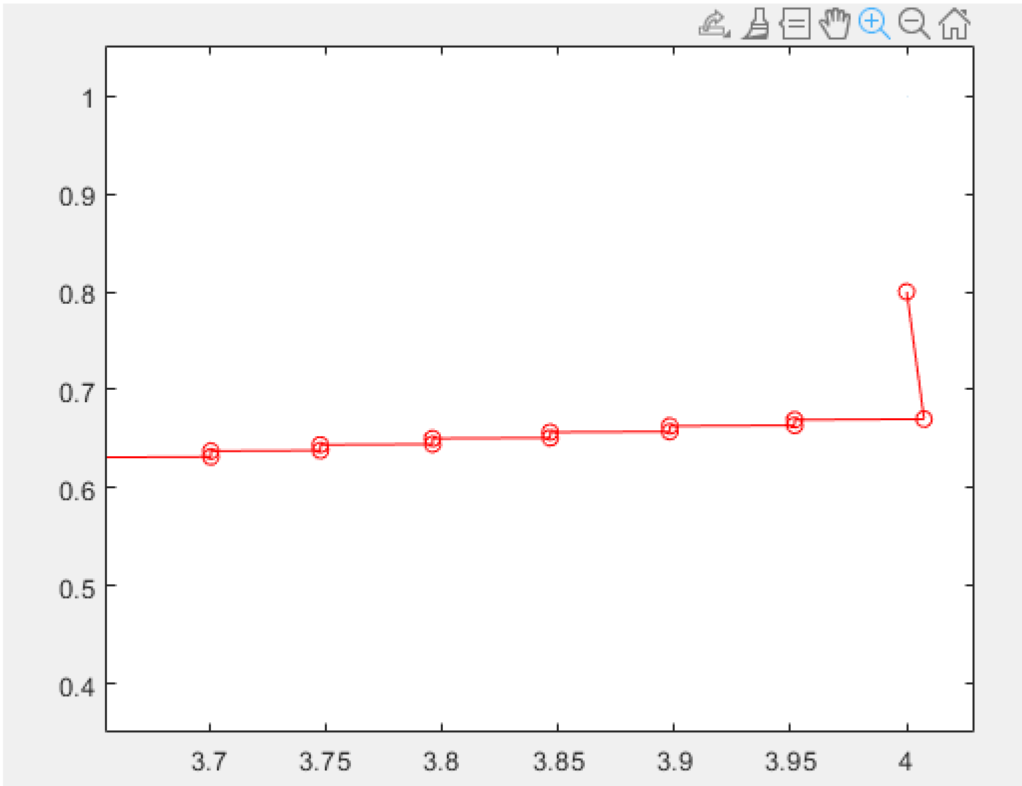

Figure 19 result of task 6b

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.

7.2. 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
```

8. Task 6 part three

Task 6

Realize the Newton's method. Comment the results.

Figure 20 task 6c

8.1. Results

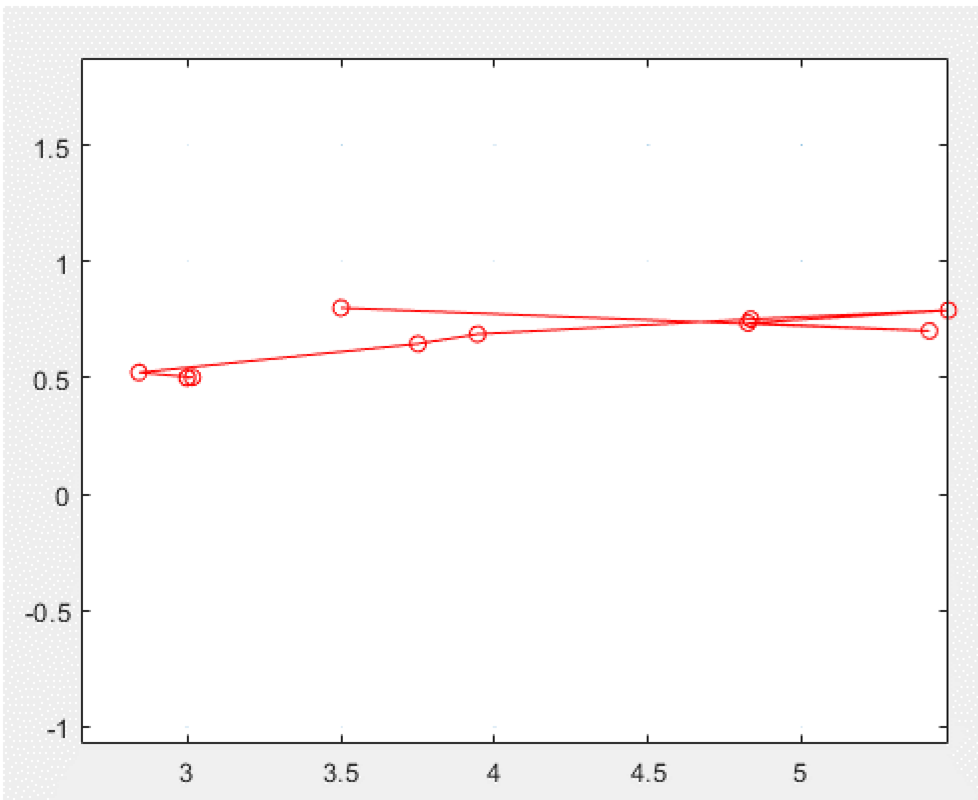

Figure 21 task 6c results

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.

8.2. 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
```

9. Control work

9.1. Task 1

① $f(x) = \frac{1}{2}(x \cdot y)\begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix} + (3\ \ 0)\begin{pmatrix}x\\ y\end{pmatrix} =$

$= \frac{1}{2}(2x+y \quad x+2y)\begin{pmatrix}x\\ y\end{pmatrix} + 3x =$

$= \frac{1}{2}(2x^2+yx+xy+2y^2)+3x = x^2+xy+y^2+3x$

$|A-\lambda I| = 0$

$\begin{vmatrix}2-\lambda & 1\\ 1 & 2-\lambda\end{vmatrix} = (2-\lambda)\cdot(2-\lambda)-1 = (\lambda-1)(\lambda-3) = 0$

$\gamma_0 = \frac{2}{\lambda_0+\lambda_*} = \frac{2}{1+3} = \frac{2}{4} = \frac{1}{2}$

$X_1 = X_0 - \gamma_0 \nabla f(X)$

$\nabla f(X) = A \cdot X_0 = \begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}\begin{pmatrix}1\\ 1\end{pmatrix} = \begin{pmatrix}3\\ 3\end{pmatrix}$

$x_1 = \begin{pmatrix}1\\ 1\end{pmatrix} - \frac{1}{2}\cdot\begin{pmatrix}3\\ 3\end{pmatrix} = \begin{pmatrix}-0,5\\ -0,5\end{pmatrix}$

Ans.: $\gamma_0 = \frac{1}{2}$. Minimum would not be reached, because the minimum of this ~~fun~~ ~~kvadratic~~ quadratic function is 0 at X value $\begin{pmatrix}0\\ 0\end{pmatrix}$

9.2. Task 2

(2) $A = \begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}$

$$f(x) = \frac{1}{2}\begin{pmatrix} x & y \end{pmatrix}\begin{pmatrix} 2 & -1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2x - y & -x + y \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} =$$

$$= \frac{1}{2}\left(2x^2 - yx - xy + y^2\right)$$

$$x_1 = x_0 - \gamma S = \begin{pmatrix} 1 \\ 3 \end{pmatrix} - \gamma \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 3-\gamma \end{pmatrix}$$

$$f(x_1) = \frac{2 \cdot 1^2 - \overset{6+2\gamma}{\cancel{\quad}} + (3-\gamma)^2}{2} = \frac{2 - 6 + 2\gamma + 9 - 6\gamma + \gamma^2}{2} =$$

$$= -2 - 2\gamma + 4{,}5 + \frac{\gamma^2}{2}$$

$$f'(x_1) = \gamma - 2 = 0$$

$$\gamma = 2 \quad \Rightarrow \quad x_1 = \begin{pmatrix} 1 \\ 3-2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$f(x_1) = \frac{2 - 1 - 1 + 1}{2} = \frac{1}{2}$$

Ans.: Minimum value: $f(X_1) = \frac{1}{2}$

Step multiplier value: $\gamma = 2$

9.3. Task 3

(3.) $A = \begin{pmatrix}1 & 0\\ 0 & 4\end{pmatrix}$

$$X_1 = X_0 - H^{-1} \cdot \nabla f(X_0)$$

$$\nabla f(X_0) = A \cdot X_0 = \begin{pmatrix}1 & 0\\ 0 & 4\end{pmatrix}\begin{pmatrix}2\\ -2\end{pmatrix} = \begin{pmatrix}2\\ -8\end{pmatrix}$$

$$H = A$$

$$H^{-1} = A^{-1}$$

$$A^{-1} = \frac{1}{|A|}\begin{pmatrix}a_{22} & -a_{12}\\ -a_{21} & a_{11}\end{pmatrix} = \frac{1}{4}\begin{pmatrix}4 & 0\\ 0 & 1\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & \frac{1}{4}\end{pmatrix}$$

$$X_1 = \begin{pmatrix}2\\ -2\end{pmatrix} - \begin{pmatrix}1 & 0\\ 0 & \frac{1}{4}\end{pmatrix}\begin{pmatrix}2\\ -8\end{pmatrix} = \begin{pmatrix}2\\ -2\end{pmatrix} - \begin{pmatrix}2\\ -2\end{pmatrix} = \begin{pmatrix}0\\ 0\end{pmatrix}$$

Ans.: $X_1 = 0$. The minimum is reach, because $f(x)$ is a quadratic function, therefore its minimum is 0.

9.4. Task 4

4.

$$S_0 = z_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$S_1 = z_1 - \left(\frac{z_1^t \cdot A \cdot S_0}{S_0^T \cdot A \cdot S_0} \right) \cdot S_0$$

$$z_1^T \cdot A \cdot S = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}^T \cdot \begin{pmatrix} 1 & 0 & 3 \\ 0 & 2 & 0 \\ 3 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = (0\ 2\ 0) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 0$$

$$S_1 = z_1 - \frac{0}{S_0^T \cdot A \cdot S_0} \cdot S_0 = z_1 - 0 \cdot S_0 = z_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$$S_2 = z_2 - \left(\frac{z_2^T \cdot A \cdot S_0}{S_0^T \cdot A \cdot S_0} \right) \cdot S_0 - \left(\frac{z_2^T \cdot A \cdot S_1}{S_1^T \cdot A \cdot S_1} \right) \cdot S_1 =$$

$$z_2^T \cdot A \cdot S_0 = (0\,0\,1) \begin{pmatrix} 1 & 0 & 3 \\ 0 & 2 & 0 \\ 3 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = (3\,0\,2) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 3$$

$$S_0^T \cdot A \cdot S_0 = (1\,0\,0) \cdot A \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 1$$

$$z_2^T \cdot A \cdot S_1 = (3\,0\,2) \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 0$$

$$S_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix} - 0 = \begin{pmatrix} -3 \\ 0 \\ 1 \end{pmatrix}$$

Ans.: $S_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ $S_1 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ $S_2 = \begin{pmatrix} -3 \\ 0 \\ 1 \end{pmatrix}$