CSI 747 – HW 8 – Submission

Jayasurya Kanukurthy – G00863457 – jkanukur@gmu.edu

Answer 1.a: The Newton based primal-dual method with primal and dual regularization for minimizing : 2*x1-3*x2, s.t. x1^2+x2^2 = 25, is as follows:

```
% Primal-Dual Newton Method Implementation of Problem 1
clear all; clc;
n=0;
x1 = 1;
x2 = 2;
y1 = 0.3;

del_x1=0;
del_x2=0;
del_y1=0;

epsilon = 0.001;
lambda = 2;
beta = 0.12;

jacobian_g = [2*x1, 2*x2];

while(norm(jacobian_g,2) >= epsilon)
    RHS = [-2+2*x1*y1; 3 + 2*x2*y1; -x1^2-x2^2+25];
    Del2_L = [-2*y1 , 0; 0, -2*y1];
    while all(eig(Del2_L)<=0)           %Regularization
        Del2_L = Del2_L + lambda * eye(length(Del2_L));
    end
    jacobian_g = [2*x1, 2*x2];
    LHS = [Del2_L, -jacobian_g'];
    [rows,cols] = size(jacobian_g);                     %Used to check if grad_g
has full rank
    leng = length(LHS);
    if (rank(jacobian_g) < min(rows,cols))              %If no full rank,
Secondary Regularization
        last_row = [jacobian_g, beta*eye(length(jacobian_g(1,:)))];
        LHS = [LHS;last_row];
    else
        last_row = [jacobian_g, zeros(rows,leng-cols)];
        LHS = [LHS;last_row];
    end

    result = LHS\RHS;

    del_x1 = result(1);
    del_x2 = result(2);
    del_y1 = result(3);

    x1 = x1+del_x1;
    x2 = x2+del_x2;
```

```
    y1 = y1 + del_y1;

    jacobian_g = [2*del_x1, 2*del_x2];          %Calculate new Jacobian of g
for finding the norm
    n = n+1;
    fprintf('Iteration= %d; x1 = %f; x2 = %f; Constraint Value = %f\n', n,
x1, x2, x1^2 + x2^2 - 25);
end
output_args = [x1;x2];
```

<u>Output</u>:

Iteration= 1; x1 = 1.000000; x2 = 7.000000; Function = -7.000000; Constraint = 25.000000
Iteration= 2; x1 = -0.440000; x2 = 5.420000; Function = -5.420000; Constraint = 4.570000
Iteration= 3; x1 = -2.388906; x2 = 4.840200; Function = -4.840200; Constraint = 4.134403
Iteration= 4; x1 = -2.864214; x2 = 4.178519; Function = -4.178519; Constraint = 0.663739
Iteration= 5; x1 = -2.773020; x2 = 4.161606; Function = -4.161606; Constraint = 0.008602
Iteration= 6; x1 = -2.773501; x2 = 4.160251; Function = -4.160251; Constraint = 0.000002
Iteration= 7; x1 = -2.773501; x2 = 4.160251; Function = -4.160251; Constraint = 0.000000

Answer 1.b: The Implementation is as follows:

```
% Primal-Dual Newton Method Implementation of Problem 1
clear all; clc;
n=0;
x1 = 2;
x2 = 3;
y1 = 0;

del_x1=0;
del_x2=0;
del_y1=0;

epsilon = 0.001;
lambda = 2;
beta = 0.12;

grad = [6*x1, 2*x2];

while(norm(grad,2) >= epsilon)
    RHS = [-2*x1-2*x2+6*x1*y1;-2*x1-2*x2+2*x2*y1;-3*x1^2-x2^2+9];
    Del2_L = [2-6*y1 , 2; 2, 2-2*y1];
    while all(eig(Del2_L)<=0)           %Regularization
        Del2_L = Del2_L + lambda * eye(length(Del2_L));
    end
    jacobian_g = [6*x1, 2*x2];
    LHS = [Del2_L, -jacobian_g'];
```

```
    [rows,cols] = size(jacobian_g);                     %Used to check if grad_g
has full rank
    leng = length(LHS);
    if (rank(jacobian_g) < min(rows,cols))              %If no full rank,
Secondary Regularization
        last_row = [jacobian_g, beta*eye(length(jacobian_g(1,:)))];
        LHS = [LHS;last_row];
    else
        last_row = [jacobian_g, zeros(rows,leng-cols)];
        LHS = [LHS;last_row];
    end

    result = LHS\RHS;

    del_x1 = result(1);
    del_x2 = result(2);
    del_y1 = result(3);

    x1 = x1+del_x1;
    x2 = x2+del_x2;
    y1 = y1 + del_y1;

    grad = [6*del_x1, 2*del_x2];           %Calculate new Jacobian of g for
finding the norm
    n = n+1;
    fprintf('Iteration= %d; x1 = %f; x2 = %f; Constraint Value = %f\n', n,
x1, x2, -(-3*x1^2-x2^2+9));

end
output_args = [x1;x2];
```

Output:

Iteration= 1; x1 = 5.000000; x2 = -5.000000; Function = 0.000000; Constraint = 91.000000
Iteration= 2; x1 = 2.725000; x2 = -2.725000; Function = 0.000000; Constraint = 20.702500
Iteration= 3; x1 = 1.775344; x2 = -1.775344; Function = 0.000000; Constraint = 3.607386
Iteration= 4; x1 = 1.521352; x2 = -1.521352; Function = 0.000000; Constraint = 0.258048
Iteration= 5; x1 = 1.500150; x2 = -1.500150; Function = 0.000000; Constraint = 0.001798
Iteration= 6; x1 = 1.500000; x2 = -1.500000; Function = 0.000000; Constraint = 0.000000

Answer 1.c. : The Newton primal-dual method implementation is as follows:

```
% Primal-Dual Newton Method Implementation of Problem 1 (c)
clear all; clc;
n=0;
x1 = 0.1;
x2 = 0.2;
x3 = 0.5;
x4 = 0.8;
y1 = -0.2;
y2 = 0.1;
```

```
del_x1=0;
del_x2=0;
del_x3=0;
del_x4=0;
del_y1=0;
del_y2=0;

epsilon = 0.001;
lambda = 2;
beta = 0.12;

grad = [2*x1, 2*x2, 2*x3, 2*x4;1, 1, 2, 3];

while(norm(grad,2) >= epsilon)
    RHS = [ -9*x1^2+2*x1*y1+y2;
            -6*x2+2*x2*y1+y2;
            -3*x3^2+2*x3*y1+2*y2;
            -3*x4+2*x4*y1+3*y2;
            -x1^2-x2^2-x3^2-x4^2+4;
            -x1-x2-2*x3-3*x4+1];

    Del2_L = [  18*x1-2*y1   0    0    0;
                0   12*x1-2*y1   0    0;
                0    0    6*x3-2*y1    0;
                0    0    0    6*x4-2*y1];


    while all(eig(Del2_L)<=0)            %Regularization
        Del2_L = Del2_L + lambda * eye(length(Del2_L));
    end
    jacobian_g = [2*x1, 2*x2, 2*x3, 2*x4;1, 1, 2, 3];
    LHS = [Del2_L, -jacobian_g'];
    [rows,cols] = size(jacobian_g);                 %Used to check if grad_g
has full rank
    leng = length(LHS);
    if (rank(jacobian_g) < min(rows,cols))          %If no full rank,
Secondary Regularization
        last_row = [jacobian_g, beta*eye(length(jacobian_g(1,:)))];
        LHS = [LHS;last_row];
    else
        last_row = [jacobian_g, zeros(rows,leng-cols)];
        LHS = [LHS;last_row];
    end

    result = LHS\RHS;

    del_x1 = result(1);
    del_x2 = result(2);
    del_x3 = result(3);
    del_x4 = result(4);
    del_y1 = result(5);
    del_y2 = result(6);
```

```
    x1 = x1+del_x1;
    x2 = x2+del_x2;
    x3 = x3+del_x3;
    x4 = x4+del_x4;
    y1 = y1 + del_y1;
    y2 = y2 + del_y2;

    grad = [x1^2+x2^2+x3^2+x4^2-4;x1+x2+2*x3+3*x4-1];
    n = n+1;
    fprintf('Iteration= %d; x1 = %f; x2 = %f; x3 = %f; x4 = %f; Function = 
%f; Constraint1 = %f; Constraint2 = %f\n', n, x1, x2, x3, x4, 
3*x1^3+2*x2^3+x3^3+x4^3, (x1^2+x2^2+x3^2+x4^2-4),(x1+x2+2*x3+3*x4-1));
end
output_args = [x1;x2];
```

Output:

Iteration= 1; x1 = -11.861556; x2 = -4.561118; x3 = 2.330018; x4 = 4.254213; Function = -5106.772549; Constraint1 = 181.027626; Constraint2 = 0.000000
Iteration= 2; x1 = -5.383180; x2 = -3.995224; x3 = 4.502994; x4 = 0.457472; Function = -504.130697; Constraint1 = 61.426671; Constraint2 = 0.000000
Iteration= 3; x1 = -2.530476; x2 = -3.602954; x3 = 1.336093; x4 = 1.487081; Function = -136.478503; Constraint1 = 19.381144; Constraint2 = 0.000000
Iteration= 4; x1 = -0.839374; x2 = -2.965798; x3 = -4.354351; x4 = 4.504625; Function = -45.102084; Constraint1 = 44.752527; Constraint2 = 0.000000
Iteration= 5; x1 = -0.023936; x2 = -2.676232; x3 = -1.748307; x4 = 2.398927; Function = -29.873921; Constraint1 = 11.974219; Constraint2 = 0.000000
Iteration= 6; x1 = 1.252897; x2 = -3.000964; x3 = 0.063224; x4 = 0.873873; Function = -47.484279; Constraint1 = 7.343189; Constraint2 = 0.000000
Iteration= 7; x1 = 0.675441; x2 = -2.026612; x3 = -0.112970; x4 = 0.859037; Function = -15.090293; Constraint1 = 1.314083; Constraint2 = -0.000000
Iteration= 8; x1 = 0.499481; x2 = -2.118356; x3 = 1.054921; x4 = 0.169678; Function = -17.459267; Constraint1 = 1.878562; Constraint2 = 0.000000
Iteration= 9; x1 = 0.157876; x2 = -2.264080; x3 = -0.118272; x4 = 1.114249; Function = -21.818076; Constraint1 = 2.406525; Constraint2 = 0.000000
Iteration= 10; x1 = 0.403061; x2 = -2.008009; x3 = 0.434540; x4 = 0.578622; Function = -15.720760; Constraint1 = 0.718186; Constraint2 = 0.000000
Iteration= 11; x1 = 0.614804; x2 = -1.830249; x3 = 0.181002; x4 = 0.617814; Function = -11.323074; Constraint1 = 0.142251; Constraint2 = 0.000000
Iteration= 12; x1 = 0.527620; x2 = -1.817040; x3 = 0.218178; x4 = 0.617688; Function = -11.311707; Constraint1 = 0.009158; Constraint2 = -0.000000
Iteration= 13; x1 = 0.512406; x2 = -1.817501; x3 = 0.221359; x4 = 0.620792; Function = -11.353832; Constraint1 = 0.000251; Constraint2 = 0.000000

Answer 2.a: The Augmented Lagrangian Implementation for Problem 2 (a) in MATLAB is as follows:

```
function [ output_args ] = AL1( x,y,k )

    f = @(x) 2*x(1)-3*x(2);
    g = @(x) x(1)^2+x(2)^2-25;

    output_args = f(x)-y'*g(x)+0.5*k*(norm(g(x))^2);

end


function [ output_args ] = AL1_Gradient( x,y,k )

    grad_f = [2;-3];
    g = @(x) x(1)^2+x(2)^2-25;
    Grad_g = @(x) [2*x(1);2*x(2)];

    output_args =   grad_f - y*Grad_g(x)+k*Grad_g(x)*g(x);

end

function [ output_args ] = AL1_Hessian( x,y,k )
        x1 = x(1);
        x2 = x(2);
        y1 = y(1);
        output_args =[ -2*y1 + k*(x1^2 + x2^2 - 25)*2 + k*4*x1^2,
k*2*(x2)*2*x1;
                        k*2*(x1)*2*x2, -2*y1 + k*(x1^2 + x2^2 - 25)*2 +
k*4*x2^2];
end

clear all; clc;
epsilon = 0.01;
x = [1;7];
y = 0;
k = 15;

g = @(xv) xv(1)^2+xv(2)^2-25;


while norm(g(x)) > epsilon
    newton_steps = 0;
    eta = 0.1;
    %Implementing Unconstrained Newton's Method
    while norm(AL1_Gradient(x,y,k)) >= max(epsilon,0.2*g(x))
        %Gradient of Augmented Lagrangian
        Gradient = AL1_Gradient(x,y,k);

        %Hessian of Augmented Lagrangian
        Hessian = AL1_Hessian(x,y,k);
```

```
        lambda = 0.00001;
        %Regularization: Checking for positive definiteness
        while min(eig( Hessian + lambda*eye(length(Hessian)))) <= 0
            lambda = 10*lambda;
        end
        %Hessian = Hessian + lambda*eye(length(Hessian));
        steps = (Hessian + lambda*eye(length(Hessian)))\-(Gradient);
        alpha = 1;
        %Armijo rule
        while (AL1(x+alpha*steps,y,k)-AL1(x,y,k)) >=
eta*alpha*AL1_Gradient(x,y,k)'*steps
            alpha = alpha/2;
        end

        x = x+alpha*steps;
        newton_steps = newton_steps+1;
    end

    fprintf('x1 = %f; x2 = %f; Newton Steps = %d; Constraint Violation =
%f\n', x(1), x(2), newton_steps, g(x));
    y = y - k *g(x);

end
```

<u>Output</u>:

x1 = -2.774811; x2 = 4.162265; Newton Steps = 19; Constraint Violation = 0.024028
x1 = -2.773491; x2 = 4.160261; Newton Steps = 1; Constraint Violation = 0.000026

Answer 2.b: The Implementation of Augmented Lagrangian Method for Problem 2.b is as follows:

```
function [ output_args ] = AL2( x,y,k )
    x1 = x(1);
    x2 = x(2);
    y1 = y(1);
    output_args = x1^2+2*x1*x2+x2^2-y1*(3*x1^2+x2^2-9)+k/2*norm(3*x1^2+x2^2-
9)^2;

end

function [ output_args ] = AL2_Gradient( x,y,k )
    x1 = x(1);
    x2 = x(2);
    y1 = y(1);
    output_args=    [2*x1 + 2*x2 - 6*y1*x1 + k*(3*x1^2 + x2^2 - 9)*6*x1;
                    2*x1 + 2*x2 - 2*y1*x2 + k*(3*x1^2 + x2^2 - 9)*2*x2];
```

```
end


function [ output_args ] = AL2_Hessian( x,y,k )
        x1 = x(1);
        x2 = x(2);
        y1 = y(1);
        output_args = [ 2 - 6*y1 + k*(3*x1^2 + x2^2 - 9)*6 + k*6*x1*6*x1,2 +
k*(2*x2)*6*x1;
                        2 + k*(6*x1)*2*x2,2 - 2*y1 + k*(3*x1^2 + x2^2 - 9)*2
+ k*(2*x2)*2*x2];
end


clear all; clc;
epsilon = 0.001;

y = 0;
x =[-2.5;2.5];

k = 1;
g = @(x) 3*x(1)^2+x(2)^2-9;

while norm(g(x)) > epsilon
    newton_steps = 0;
    eta = 0.1;
    %Implementing Unconstrained Newton's Method
    while norm(AL2_Gradient(x,y,k)) >= max(epsilon,0.2*g(x))
        %Gradient of Augmented Lagrangian
        Gradient = AL2_Gradient(x,y,k);

        %Hessian of Augmented Lagrangian
        Hessian = AL2_Hessian(x,y,k);

        lambda = 0.00001;
        %Regularization: Checking for positive definiteness
        while min(eig( Hessian + lambda*eye(length(Hessian)))) <= 0
            lambda = 10*lambda;
        end
        %Hessian = Hessian + lambda*eye(length(Hessian));
        steps = (Hessian + lambda*eye(length(Hessian)))\-(Gradient);
        alpha = 1;
        %Armijo rule
        while (AL2(x+alpha*steps,y,k)-AL1(x,y,k)) >=
eta*alpha*AL2_Gradient(x,y,k)'*steps
            alpha = alpha/2;
        end

        x = x+alpha*steps;
        newton_steps = newton_steps+1;
    end

    fprintf('x1 = %f; x2 = %f; Newton Steps = %d; Constraint Violation =
%f\n', x(1), x(2), newton_steps, g(x));
    y = y - k *g(x);
```

```
end
```

Output:

x1 = -1.500000; x2 = 1.500000; Newton Steps = 5; Constraint Violation = 0.000000

Answer 2.c:

The Implementation of Augmented Lagrangian Method for Problem 2.c is as follows:

```
function [ output_args ] = AL3( x,y,k )
    x1 = x(1); x2 = x(2); x3 = x(3); x4  = x(4);
    y1 = y(1);y2 = y(2);

    g = [x1^2+x2^2+x3^2+x4^2-4;x1+x2+2*x3+3*x4-1];
    output_args = 3*x1^3+2*x2^3+x3^3+x4^3-[y1 y2]*g+k/2*norm(g)^2;

end


function [ output_args ] = AL3_Gradient(  x,y,k )
    x1 = x(1); x2 = x(2); x3 = x(3); x4  = x(4);
    y1 = y(1);y2 = y(2);
    g1 = x1^2+x2^2+x3^2+x4^2-4;
    g2 = x1+x2+2*x3+3*x4-1;
    output_args=    [9*x1^2-2*x1*y1-y2+2*k*x1*g1+k*g2;
                     6*x2^2-2*x2*y1-y2+2*k*x2*g1+k*g2;
                     3*x3^2-2*x3*y1-2*y2+2*k*x3*g1+2*k*g2;
                     3*x4^2-2*x4*y1-3*y2+2*k*x4*g1+3*k*g2];
end


function [ output_args ] = AL3_Hessian(  x,y,k )
    x1 = x(1); x2 = x(2); x3 = x(3); x4  = x(4);
    y1 = y(1);y2 = y(2);
    output_args = [ 18*x1 - 2*y1 + k* (x1^2 + x2^2 + x3^2 + x4^2 - 4)*2 +
k*2*x1*2*x1 + k,k*2* (x2)*2*x1 + k,  k*2* (x3)*2*x1 + k*2,k*2* (x4)*2*x1 +
k*3;
k*2* (x1)*2*x2 + k, 12*x2 - 2*y1 + k* (x1^2 + x2^2 + x3^2 + x4^2 - 4)*2 +
k*2*x2*2*x2 + k, k*2* (x3)*2*x2 + k*2, k*2* (x4)*2*x2 + k*3;
k*2* (x1)*2*x3 + k*2,k*2* (x2)*2*x3 + k*2, 6*x3 - 2*y1 + k* (x1^2 + x2^2 +
x3^2 + x4^2 - 4)*2 + k*2*x3*2*x3 + k*2*2,k*2* (x4)*2*x3 + k*2*3;
k*2* (x1)*2*x4 + k*3,k*2* (x2)*2*x4 + k*3,k*2* (x3)*2*x4 + k*3*2,6*x4 - 2*y1
+ k* (x1^2 + x2^2 + x3^2 + x4^2 - 4)*2 + k*2*x4*2*x4 + k*3*3];

end
clear all; clc;
epsilon = 0.01;
```

```
y = [0;0];
x = [1;2;3;4];
k = 100;
g = @(x) [x(1)^2+x(2)^2+x(3)^2+x(4)^2-4; x(1)+x(2)+2*x(3)+3*x(4)-1];

while norm(g(x)) > epsilon
    newton_steps = 0;
    eta = 0.1;
    %Implementing Unconstrained Newton's Method
    while norm(AL3_Gradient(x,y,k)) >= max(epsilon,0.2*g(x))
        %Gradient of Augmented Lagrangian
        Gradient = AL3_Gradient(x,y,k);

        %Hessian of Augmented Lagrangian
        Hessian = AL3_Hessian(x,y,k);

        lambda = 0.00001;
        %Regularization: Checking for positive definiteness
        while min(eig( Hessian + lambda*eye(length(Hessian)))) <= 0
            lambda = 10*lambda;
        end
        %Hessian = Hessian + lambda*eye(length(Hessian));
        steps = (Hessian + lambda*eye(length(Hessian)))\-(Gradient);
        alpha = 1;
        %Armijo rule
        while (AL3(x+alpha*steps,y,k)-AL1(x,y,k)) >=
eta*alpha*AL3_Gradient(x,y,k)'*steps
            alpha = alpha/2;
        end

        x = x+alpha*steps;
        newton_steps = newton_steps+1;
    end

    fprintf('x1 = %f; x2 = %f; x3 = %f; x4 = %f; Newton Steps = %d;
Constraint1 = %f;Constraint2 = %f\n', x(1), x(2),x(3),x(4), newton_steps,
g(x));
    y = y - k *g(x);
end
```

Output:

x1 = -1.870848; x2 = 0.207074; x3 = 0.414149; x4 = 0.600589; Newton Steps = 14; Constraint1 = 0.075179;Constraint2 = -0.033708
x1 = -1.849036; x2 = 0.208028; x3 = 0.416056; x4 = 0.603051; Newton Steps = 2; Constraint1 = -0.001018;Constraint2 = 0.000256