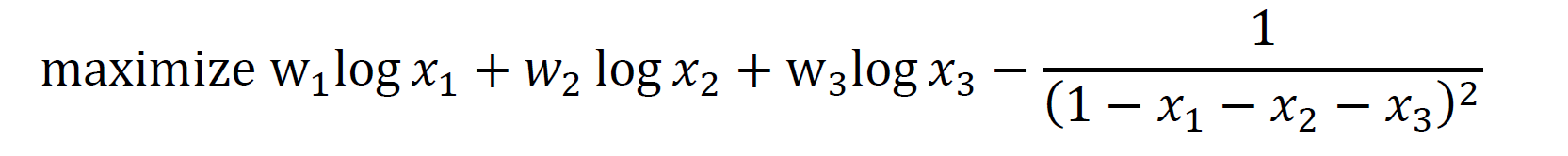
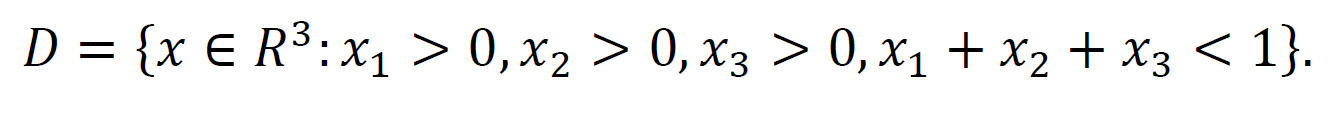
**Programming Assignment 2, Due: Part 1:**

**P.1**.: Solve the following problem using backtracking line search (Armijo’s rule) in Matlab:

  
Here *w1* , *w2* , *w3* are constants and we have :

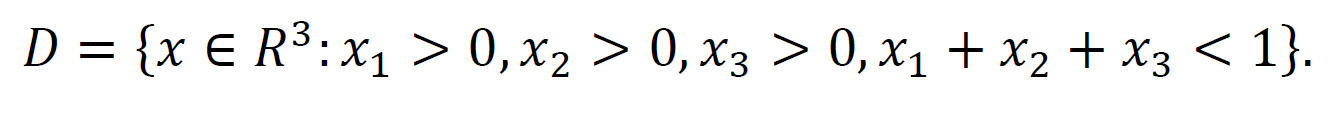


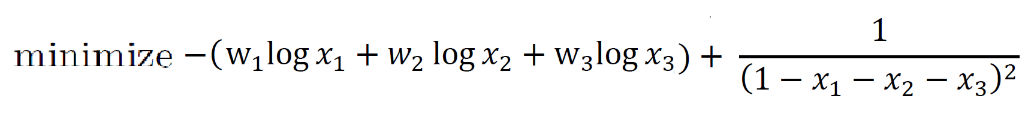
The following instructions apply to both parts 1 and 2 (part 2 is on the next page).

1. The stopping criterion you used, i.e., the value of in the criterion .
2. The optimal solution produced by your program for the following two cases: w1=1 ,w2 =1 ,w3=1 and w1=1 ,w2 =2 ,w3=3.

In addition, present a solution for another case where the constants are chosen by you.

**Solution** :

At first we should turn the primal problem into standard form ,In this case the objective function would be convex . thus we have :



We use *gradient method* and check the opt. solution with *Newton’s method* (just for this part) .

For *gradient method ,* The figures show the function values and step lengths versus iteration

number for an example with We used *α* = 0.01, *β* = 0.5 and exit condition and exit condition

10 – 6 and we guess initial vector *x*0 = 0.1 .





And optimal point ([x1 x2 x3]) and required iteration number (k) is in the below:

x =

[ 0.1239 0.1239 0.1239 ]

k = 19

The following is a Matlab implementation (for *gradient method*):

----------------------------------------------------------------**clear all;**

**clc;**

**ALPHA = 0.01;**

**BETA = 0.5;**

**MAXITERS = 1000;**

**GRADTOL = 1e-6;**

**W=[1 ;1 ;1];**

**A = ones(3,1);**

**x = zeros(3,1)+0.1;**

**k=1;%counter for getting to optimal value that fills our matrix (fk) up to that number**

**for iter = 1:MAXITERS**

**val = -W'\*log(x)+1./((1-A'\*x).^2);**

**fk(k)=val ;**

**d=1./x;**

**grad = -W.\*d+2\*A\*(1./(1-A'\*x).^3);**

**if norm(grad) < GRADTOL, break; end;**

**k=k+1;**

**v = -grad;**

**fprime = grad'\*v;**

**t = 1; while ((sum(x+t\*v) >= 1) | (min(x+t\*v) <= 0)),**

**t = BETA\*t;**

**end;**

**while ( -W'\*log(x+t\*v)+1./((1-A'\*(x+t\*v)).^2) > ...**

**val + ALPHA\*t\*fprime )**

**t = BETA\*t;**

**end;**

**tk(k)=t;**

**x = x+t\*v;**

**end;**

**semilogy(1:k,fk(1:k)-fk(k));**

**xlabel('k')**

**ylabel('fk - p\*')**

**figure**

**plot(1:k,tk,'o');**

**xlabel('k');**

**ylabel('tk');**

**x,k**

----------------------------------------------------------------

And now we check this with *Newton’s method*:

In this method the parameters are same except that we look for λ 10 – 10 and we select initial vector *x*0 = 0.3 that is worse than previous guess but for better showing convergence (because of high speed of *Newton’s method* convergence) we do this :





And optimal point ([x1 x2 x3]) and required iteration number (k) is in the below:

x =

[ 0.1239 0.1239 0.1239 ]

k = 10

That we see this method has same answer and despite the bad initial guess the number of iteration is less than before.

The following is a Matlab implementation (for *Newton’s method*):

----------------------------------------------------------------

**clear all;**

**clc;**

**ALPHA = 0.01;**

**BETA = 0.5;**

**MAXITERS = 1000;**

**NTTOL = 1e-10;**

**W=[1 ;1 ;1];**

**A = ones(3,1);**

**x = zeros(3,1)+0.3;**

**k=1;%counter for getting to optimal value that fills our matrix (fk) up to that number**

**for iter = 1:MAXITERS**

**val = -W'\*log(x)+1./((1-A'\*x).^2);**

**fk(k)=val ;**

**d=1./x;**

**grad = -W.\*d+2\*A\*(1./(1-A'\*x).^3);**

**hess = diag(d.^2)+ 6\*A\*(1./((1-A'\*x).^4))\*A';**

**v = -hess\grad;**

**fprime = grad'\*v;**

**if abs(fprime) < NTTOL, break; end;**

**k=k+1;**

**t = 1; while ((sum(x+t\*v) >= 1) | (min(x+t\*v) <= 0)),**

**t = BETA\*t;**

**end;**

**while ( -W'\*log(x+t\*v)+1./((1-A'\*(x+t\*v)).^2) > ...**

**val + ALPHA\*t\*fprime )**

**t = BETA\*t;**

**end;**

**tk(k)=t;**

**x = x+t\*v;**

**end;**

**semilogy(1:k,fk(1:k)-fk(k));**

**xlabel('k')**

**ylabel('fk - p\*')**

**figure**

**plot(1:k,tk,'o');**

**xlabel('k');**

**ylabel('tk');**

**axis([0,11,0,1.5])**

**x,k**

----------------------------------------------------------------In continue we solve the problem with *gradient method* for condition of (w1=1 ,w2 =2 ,w3=3) :

The matlab code is same (*gradient method* code with same parameters) just we change **W** into **W=[6 ;2.6 ;87].**





And optimal point ([x1 x2 x3]) and required iteration number (k) is in the below:

x =

[ 0.0772 0.1545 0.2317 ]

k = 33

----------------------------------------------------------------

In continue we solve the problem with *gradient method* for arbitrary condition of (w1=6 ,w2 =2.6 ,w3=87) :

The matlab code is same (*gradient method* code with same parameters) just we change **W** into **W=[6 ;2.6 ;87].**





And optimal point ([x1 x2 x3]) and required iteration number (k) is in the below:

x =

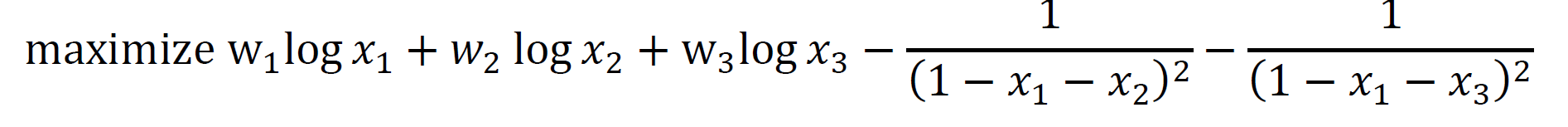
[ 0.0471 0.0204 0.6822 ]

k = 49

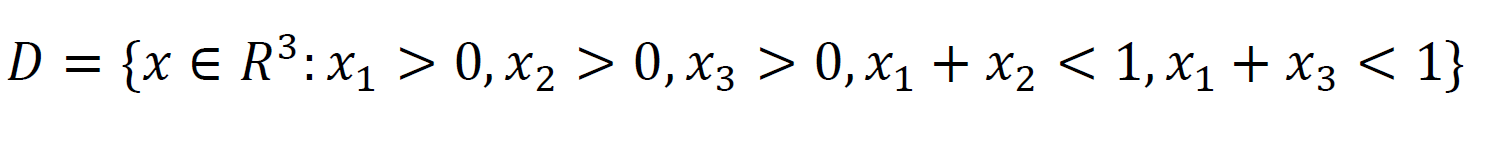
----------------------------------------------------------------

**Programming Assignment 2, Part 2:**

**P.2**.: Solve the following problem using backtracking line search (Armijo’s rule) in Matlab:

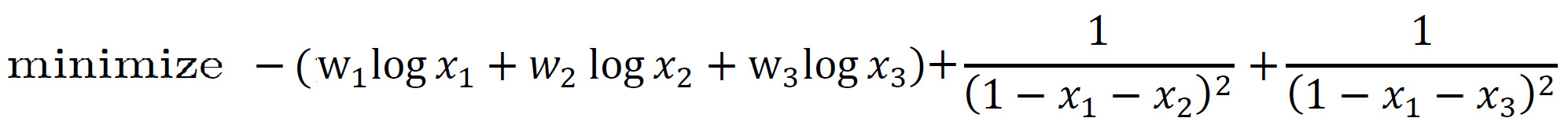


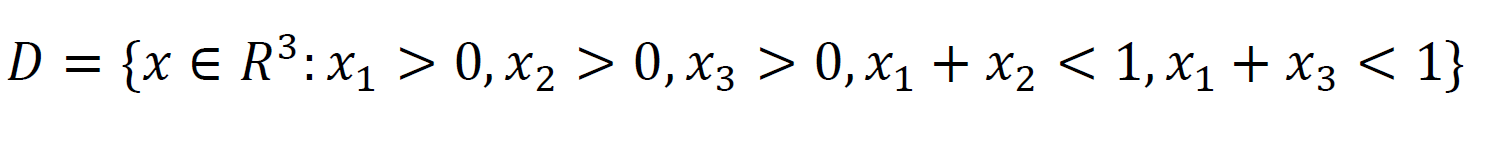
Here *w1* , *w2* , *w3* are constans and we have :



**Solution** :

At first we should turn the primal problem into standard form ,In this case the objective function would be convex . thus we have :





We use *gradient method* for condition of (w1=1 ,w2 =1 ,w3=1) and *α* = 0.01, *β* = 0.5 and exit

condition and exit condition 10 – 6 and we guess initial vector *x*0 = 0.1 .





And optimal point ([x1 x2 x3]) and required iteration number (k) is in the below:

x =

[ 0.0455 0.3247 0.7018 ]

k = 436

The following is a Matlab implementation :

----------------------------------------------------------------

**clear all;**

**clc;**

**ALPHA = 0.01;**

**BETA = 0.5;**

**MAXITERS = 1000;**

**GRADTOL = 1e-6;**

**W=[6 ;2.6 ;87];**

**A1 = [1;1;0];**

**A2 = [1;0;1];**

**x = zeros(3,1)+0.1;**

**k=1;%counter for getting to optimal value that fills our matrix (fk) up to that number**

**for iter = 1:MAXITERS**

**val = -W'\*log(x)+1./((1-A1'\*x).^2)+1./((1-A2'\*x).^2);**

**fk(k)=val ;**

**d=1./x;**

**grad = -W.\*d+2\*(A1\*(1./(1-A1'\*x).^3)+A2\*(1./(1-A2'\*x).^3));**

**if norm(grad) < GRADTOL, break; end;**

**k=k+1;**

**v = -grad;**

**fprime = grad'\*v;**

**t = 1; while ((A1'\*(x+t\*v) >= 1) | (min(x+t\*v) <= 0) | ((A2'\*(x+t\*v) >= 1) >= 1)),**

**t = BETA\*t;**

**end;**

**while ( -W'\*log(x+t\*v)+1./((1-A1'\*(x+t\*v)).^2)+1./((1-A2'\*(x+t\*v)).^2) > ...**

**val + ALPHA\*t\*fprime )**

**t = BETA\*t;**

**end;**

**tk(k)=t;**

**x = x+t\*v;**

**end;**

**semilogy(1:k,fk(1:k)-fk(k));**

**xlabel('k')**

**ylabel('fk - p\*')**

**figure**

**plot(1:k,tk,'-o');**

**xlabel('k');**

**ylabel('tk')**

**x,k**

----------------------------------------------------------------

Like previous part we try that with condition of (w1=1 ,w2 =2 ,w3=3) :

The matlab code is same (with same parameters) just we change **W** into **W=[1 ;2 ;3].**





And optimal point ([x1 x2 x3]) and required iteration number (k) is in the below:

x =

[ 0.0621 0.2821 0.3326 ]

k = 47

----------------------------------------------------------------

And for arbitrary condition of (w1=7 ,w2 =0.8 ,w3=21) :

The matlab code is same (*gradient method* code with same parameters) just we change **W** into **W=[7 ;0.8 ;21].**





And optimal point ([x1 x2 x3]) and required iteration number (k) is in the below:

x =

[ 0.1455 0.1437 0.4936 ]

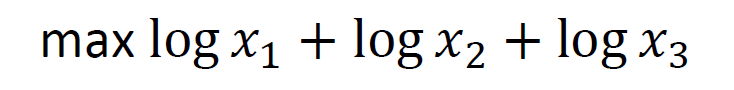
k = 126

----------------------------------------------------------------

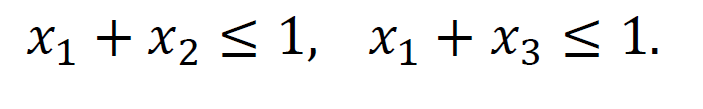
**Programming Assignment 3, Due:**

**Section 1**:

In this assignment, you will solve the following problem using the interior point method :



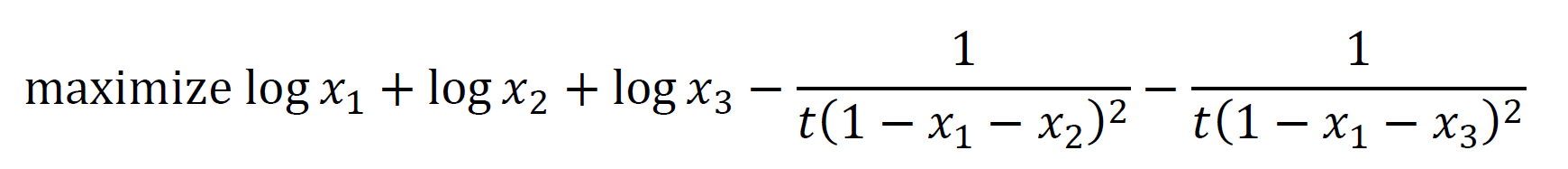
S.t.



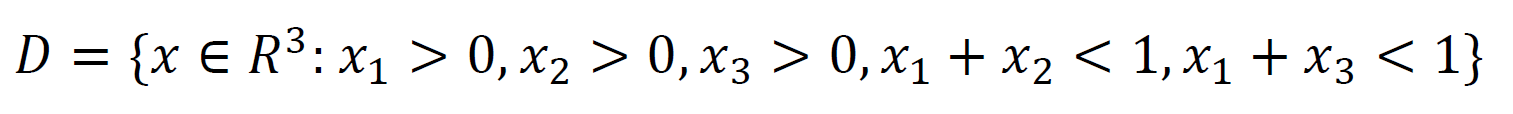
To implement the interior point method, start with *t* = 200 and use 𝛽 = 20. In other words, the sequence of values for *t* should be 1020, 10, 10, etc. The last value of *t* that you choose should be 10. Start with an initial guess of 𝑥1 *=* 𝑥2 *=* 𝑥3= 0.3.

**Section 2**:

For each of the above values of *t*, solve the following problem using backtracking line search a.k.a. Armijo’s rule (use the following parameters for the line search: 𝛼=0.25,𝛽=0.1) in Matlab:



The domain of the above objective function is given by



For each t, use the solution for the previous value of *t* as the initial guess for *x*, except for *t* = 200 when you will use the initial guess stated earlier. Please use the following stopping criterion for each value of *t*≤ 0.0001.

Determine the final value of 𝑥1, 𝑥2, 𝑥3, and the number of iterations of the gradient method that you had to execute for each value of *t*. Also, compare this answer with the number of iterations required when you directly use the gradient algorithm to solve the barrier-function formulation with *t* = 10, starting with 𝑥1 = 𝑥2 = 𝑥3 = 0.3.

**Solution section 1**:

First we turn problem into standard form like this :

minimize - ( *log* ( *x*1 ) + *log* ( *x*2 ) + *log* ( *x*3 ) )

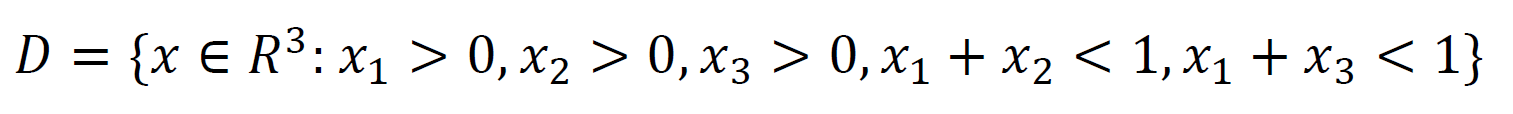
S.t.

*g*1 ( *x* ) = 1 - ( *x*1 + *x*2 ) *and g*2 ( *x* ) = 1 - ( *x*1 + *x*3 )

Then for solve this problem we use barrier method so we replace that with this problem :

minimize - ( *log* ( *x*1 ) + *log* ( *x*2 ) + *log* ( *x*3 ) ) – ( ) ( [*log* (- *g*1 ( *x* ) ) + *log* (- *g*2 ( *x* ) ) ] )

The domain of the above objective function is given by



For solve this we write matlab code and find optimal vector [ *x*1 *x*2 *x*3] and iteration number ( *k* ) for each *t .*

The following is a Matlab implementation :

----------------------------------------------------------------

**clear all;**

**clc;**

**ALPHA = 0.25;**

**BETA = 0.1;**

**MAXITERS = 400000;**

**GRADTOL = 1e-4;**

**B=20;**

**t=200;**

**A1 = [1;1;0];**

**A2 = [1;0;1];**

**k=ones(7,1);%number of iteration**

**for i=1:7**

**if t==200**

**x = zeros(3,1)+0.3;**

**end;**

**for iter = 1:MAXITERS**

**val = -sum(log(x))-(1/t)\*(log(1-A1'\*x)+log(1-A2'\*x));**

**d=1./x;**

**grad = -d+(1/t)\*(A1\*(1./(1-A1'\*x))+A2\*(1./(1-A2'\*x)));**

**if norm(grad) < GRADTOL, break; end;**

**k(i)=k(i)+1;**

**v = -grad;**

**fprime = grad'\*v;**

**tt = 1; while ((A1'\*(x+tt\*v) >= 1) | (min(x+tt\*v) <= 0) | ((A2'\*(x+tt\*v) >= 1) >= 1)),**

**tt = BETA\*tt;**

**end;**

**while ( -sum(log(x+tt\*v))-(1/t)\*(log(1-A1'\*(x+tt\*v))+log(1-A2'\*(x+tt\*v))) > ...**

**val + ALPHA\*tt\*fprime )**

**tt = BETA\*tt;**

**end;**

**x = x+tt\*v;**

**end; %The loop of computing centeral point with gradient method**

**t=t\*B;**

**end;%The loop of computing interior point**

**x,k**

----------------------------------------------------------------

And optimal vector ([ *x*1 *x*2 *x*3]) and required iteration number (*k*) is in the below:

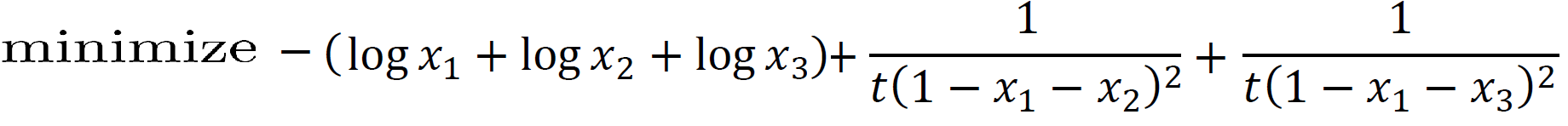
x =

[ 0.3333 0.6667 0.6667 ]

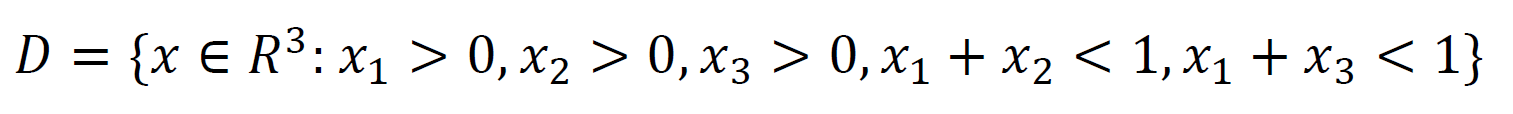
|  |  |
| --- | --- |
| k =  508 | t=  1020 |
| 23925 | 10 |
| 88286 | 10 |
| 8372 | 10 |
| 400000  (The number of iteration required for Armijo’s rule more than 400000 ) | 10 |
| 400000  (The number of iteration required for Armijo’s rule more than 400000 ) | 10 |
| 400000  (The number of iteration required for Armijo’s rule more than 400000 ) | 10 |

**Solution section 2**:

First we turn problem into standard form like this :



The domain of the above objective function is given by :



For solve this we write matlab code and find optimal vector [ *x*1 *x*2 *x*3] and iteration number ( *k* ) for each *t .*

The following is a Matlab implementation :

----------------------------------------------------------------

**clear all;**

**clc;**

**ALPHA = 0.25;**

**BETA = 0.1;**

**MAXITERS = 400000;**

**GRADTOL = 1e-4;**

**B=20;**

**t=200;**

**A1 = [1;1;0];**

**A2 = [1;0;1];**

**k=ones(7,1);%number of iteration**

**for i=1:7**

**if t==200**

**x = zeros(3,1)+0.3;**

**end;**

**for iter = 1:MAXITERS**

**val = -sum(log(x))+(1/t)\*((1./(1-A1'\*x).^2)+(1./(1-A2'\*x).^2));**

**d=1./x;**

**grad = -d+2\*(1/t)\*(A1\*(1./(1-A1'\*x).^3)+A2\*(1./(1-A2'\*x).^3));**

**if norm(grad) < GRADTOL, break; end;**

**k(i)=k(i)+1;**

**v = -grad;**

**fprime = grad'\*v;**

**tt = 1; while ((A1'\*(x+tt\*v) >= 1) | (min(x+tt\*v) <= 0) | ((A2'\*(x+tt\*v) >= 1) >= 1)),**

**tt = BETA\*tt;**

**end;**

**while ( -sum(log(x+tt\*v))+(1/t)\*((1./(1-A1'\*(x+tt\*v)).^2)+(1./(1-A2'\*(x+tt\*v)).^2)) > ...**

**val + ALPHA\*tt\*fprime )**

**tt = BETA\*tt;**

**end;**

**x = x+tt\*v;**

**end;**

**t=t\*B;**

**end;**

**x,k**

----------------------------------------------------------------

And optimal vector ([ *x*1 *x*2 *x*3]) and required iteration number (*k*) is in the below:

x =

[ 0.3332 0.6663 0.6663 ]

|  |  |
| --- | --- |
| k =  25 | t=  1020 |
| 300 | 10 |
| 372 | 10 |
| 438 | 10 |
| 2971 | 10 |
| 101 | 10 |
| 17695 | 10 |

We result from sec.1 and sec.2 that the objective function that used in sec.2 can improve itration number (because of less itration number)and be replaced with barrier function that used in sec.1 for optimization problem , because of the two both functions have approximately the same answer (error order ).