CS 791 Project
Batyr Charyyev

Question1 - (Implementation can be found on Appendix A)

$$\text{minimize} \quad \tfrac{1}{2}\, x^{T} Q\, x + q^{T} x$$

gradient: $\nabla f(x) = Q\,x + q$
gradient descent direction: $\Delta x = -\nabla f(x)$
steepest descent direction: $\Delta x_{sd} = -P^{-1}\nabla f(x)$
//P is diagonal matrix with diagonal elements same as Q
stopping criterion: $\|\nabla f(x)\|_2 \leq$ tolerance

Algorithm used to solve this unconstrained problem.

select a starting point $x \in$ dom f
do
1. Determine a descent direction Δx.
2. Line search. Choose a step size t > 0.
   - Exact Line Search
   - Backtracking Line Search
3. Update x := x + tΔx.

while !(stopping criterion is satisfied)

Exact line search: $\mathbf{t=\operatorname{argmin}_{s\geq 0} f(x+s\Delta x)}$

- t is chosen to minimize f (objective function) along the ray { x+tΔx|t≥0 }

Backtracking line search:

given a descent direction Δx for f at $x \in dom\ f$, $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$
$t := 1.$
while $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^{T} \Delta x, \quad t := \beta t$

Steepest descent algorithm:

select a starting point $x \in$ dom f
do
1. Determine a steepest descent direction Δx_{sd}
2. Line search. Choose a step size t > 0.
   - Exact Line Search
   - Backtracking Line Search
3. Update $x := x + t\Delta x_{sd}$

while !(stopping criterion is satisfied)

Graphs: Error $f(x^{(k)}) - p^\star$ versus iteration k graphs, for problem above with different staring points. Backtracking (α=0.1 and β=0.5) and exact line search comparison.
Maximum number of iteration:200
Tolerance: 1e-6;
q = [34; 6; 10; 29; 50]
Q = [10257 6272 3398 6519 8102
6272 5066 4062 7395 8503
3398 4062 8712 8400 13931
6519 7395 8400 17675 19603
8102 8503 13931 19603 27041]

a) Implementation code can be found on Appendix A.
b)

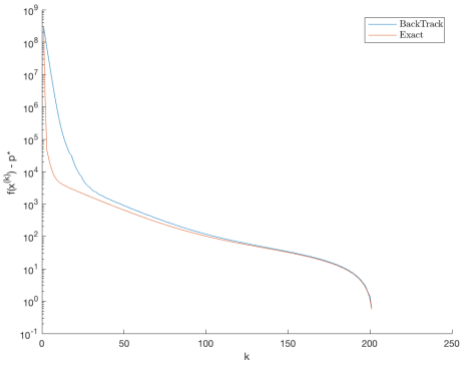


Figure-1: x[6;51;4;65;78]

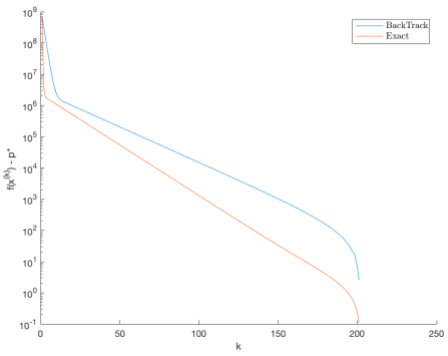


Figure-2: x[41;96;95;96;59]

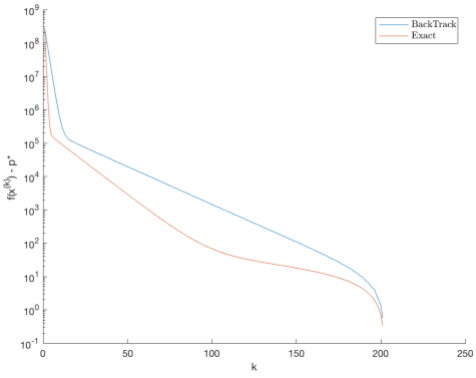


Figure-3: x[81;83;18;35;40]

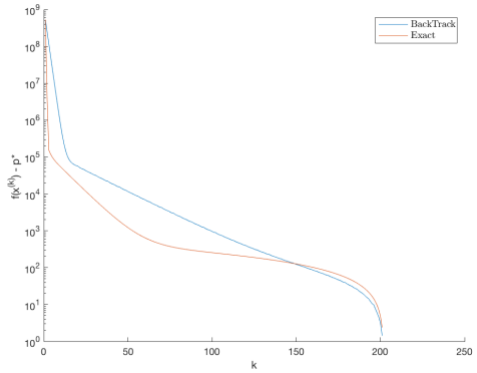


Figure-4: x[19;3;93;92;75]

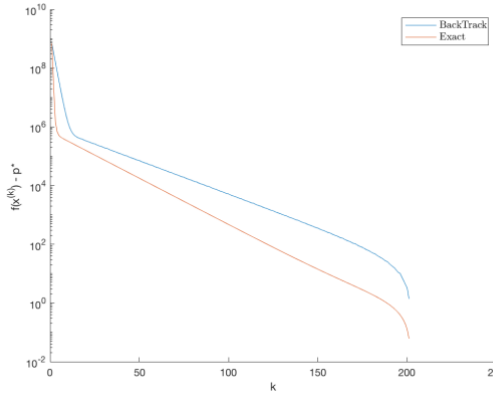


Figure-5: x[104;97;62;79;57]

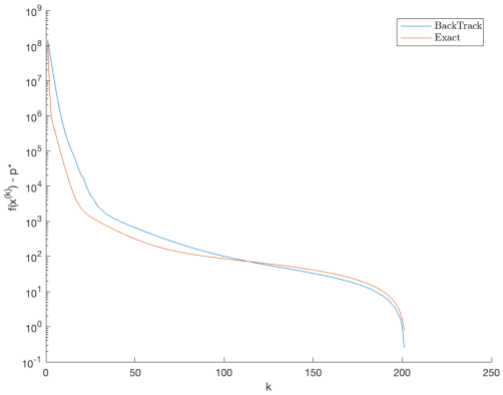


Figure-6: x[68;6;51;19;33]

c) Exact line search and backtrack line search comparison provided on Figures 1-6. Comparison of different alpha and beta on backtrack line search can be found on Table 1 for same Q, q and x = [76; 9; 9; 14; 12]. Tolerance was set to 1.

Table 1: Comparison of different alpha and beta on backtrack line search

Beta	Alpha	Number of Iteration	Optimum Value	Tolerance
0.1	0.1	30108	-0.7221	1
0.1	0.25	30019	-0.7187	1
0.1	0.4	29933	-0.7154	1
0.25	0.1	10356	-0.7129	1
0.25	0.25	9888	-0.7119	1
0.25	0.4	7855	-0.7126	1
0.5	0.1	32498	-0.7582	1
0.5	0.25	32082	-0.7471	1
0.5	0.4	22527	-0.7162	1
0.75	0.1	37321	-0.799	1
0.75	0.25	35964	-0.7785	1
0.75	0.4	26049	-0.7376	1
0.9	0.1	41195	-0.8253	1
0.9	0.25	37991	-0.7994	1
0.9	0.4	36310	-0.7765	1

From figures we can observe that exact line search usually improves the convergence of the gradient method, but the effect is not large. We varied alpha between 0.1 and 0.4 meaning that we accept a decrease in f between 1% and 40% of the prediction based on the linear extrapolation. Parameter beta was varied between 0.1(very crude search) and 0.9(less crude search).
We can observe that for fixed small beta (β =0.1) variation in total number of iterations is not large 175 (30108-29933) and again for fixed large beta (β =0.9) variation in total number of iterations is 4885. When we fix alpha and vary beta variation in total number of iterations is at most 30839.
We can conclude that beta plays more important role in convergence and our little experiment suggest that $\beta \approx 0.25$ is a good choice.
Effect of alpha and beta on convergence rate is less compared to other parameters like condition number.

d)

Table 2: Different problem size comparison

n	Backtrack: Number of Iteration	Exact: Number of Iteration	Backtrack:Optimal Value	Exact: Optimal Value
5	2897	3625	-5.7687	-5.7696
50	200001	200001	1.73E+03	1.83E+03
100	200001	200001	9.22E+03	1.05E+04
250	200001	200001	1.01E+05	1.67E+05
350	200001	200001	2.32E+05	4.82E+05
500	200001	200001	1.40E+06	2.19E+06

As shown on Table2, we used different problem sizes and we set maximum number of iteration to 200thousand. We can observe that as problem size increases in 200thousand iteration the converging value is becoming less optimal meaning that number of needed iteration is also increases to obtain specific optimal value.

e)
We set maximum number of iteration to 200thousand, tolerance to 1 and each time we created random 10x10 matrix with different condition number. Summary of results obtained for part e and f provided on Table 3.
Where: CN: condition number

GBNI: Number of iteration for gradient backtracking line search method
GENI: Number of iteration for gradient exact line search method
GBOP: Optimal value for gradient backtracking line search method
GEOP: Optimal value for gradient exact line search method
SBNI: Number of iteration for steepest descent backtracking line search method
SENI: Number of iteration for steepest descent exact line search method
SBOP: Optimal value for steepest descent backtracking line search method
SEOP: Optimal value for steepest descent exact line search method

Figure 7 shows line chart of comparison where x-axis is condition number and y-axis is number of iteration needed in log-scale

Table 3: Different condition number comparison

CN	GBNI	GENI	GBOP	GEOP	SBNI	SENI	SBOP	SEOP
483.4073	773	2169	-1.0971	-1.0974	2245	200001	-1.0974	2.76E+06
611.5327	1091	3111	-2.6651	-2.6655	1832	200001	-2.6655	1.01E+07
8.26E+02	3431	3891	-0.4669	-0.4671	2948	200001	-0.4667	2.56E+07
2.61E+03	3959	7368	-3.7177	-3.7184	9013	200001	-3.72	6.34E+06
4.24E+03	7556	11533	-6.8187	-6.8195	8592	200001	-6.821	7.14E+06
5.18E+03	12608	17214	-3.6402	-3.6415	11492	200001	-3.64	1.88E+07
1.91E+04	15308	38804	-11.355	-11.3664	39117	200001	-11.3535	4.76E+06
5.63E+04	29961	119185	-5.9716	-6.0067	130241	200001	-6.0112	4.46E+06
8.98E+05	200001	200001	-4.44E+03	-3.23E+03	200001	200001	-1.84E+03	2.51E+06

If the condition number of a set C is small, it means that the set has approximately the same width in all directions, i.e., it is nearly spherical. If the condition number is large, it means that the set is far wider in some directions than in others.
Number of iteration required to obtain given level of accuracy grows with condition number. We can see it from Table3 and Figure7. Overall gradient method is slow when condition number is large and fast when condition number is low. Gradient method strongly depends on condition number.

f)
Implementation of steepest method can be found on Appendix A. Convergence rate in steepest descent depends on matrix P. If we can identify a matrix P for which the transformed problem has moderate condition number our steepest algorithm works well. From results that we obtained we can conclude steepest descent method also depends on condition number as gradient descent method.

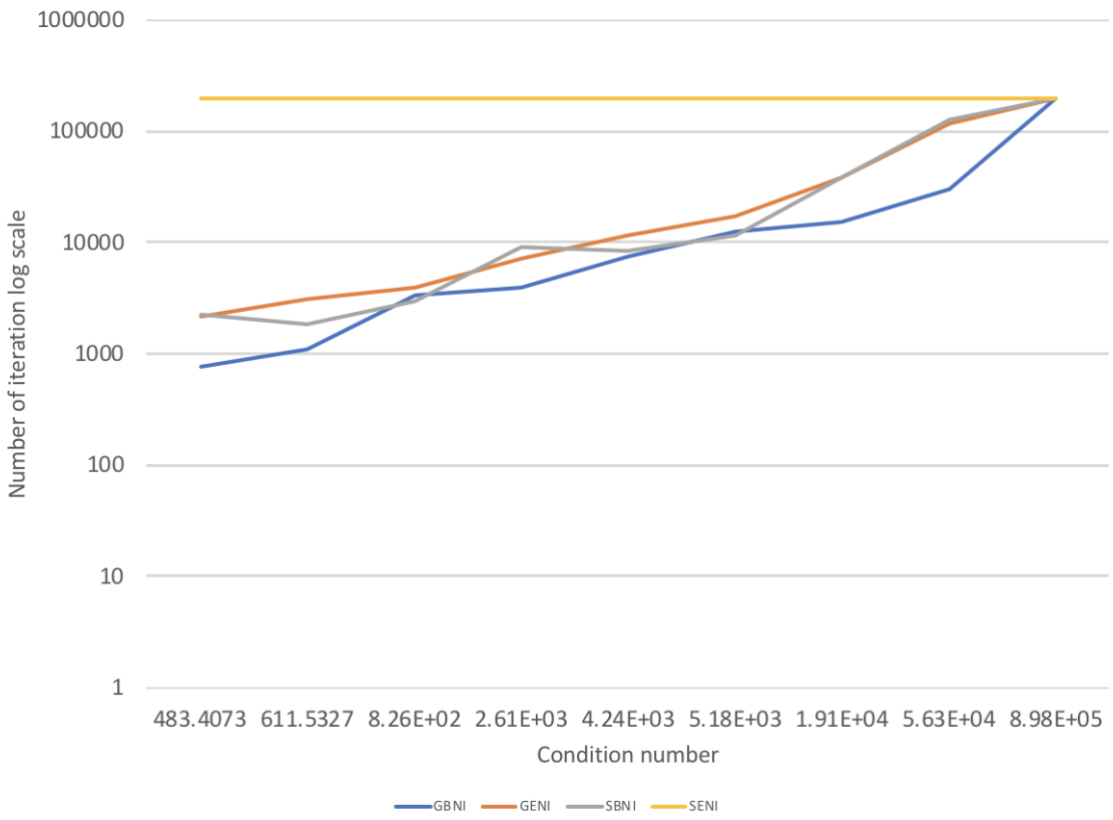


Figure-7: Condition number analyses

g)

Least squares problem is one special case of quadratic minimization problem so we can transform it to equation above.

$$\text{minimize} \quad \|Ax-b\|_2^2 = x^T(A^TA)x - 2(A^Tb)^Tx + b^Tb.$$

When we compare computational advantages, if we try to solve problem by solving system of linear equations it is costly compared to gradient method. Because taking inverse of large matrix is costly operation so for large problem Gradient descent is much faster.

One potential disadvantage of using Gradient descent or Steepest descent method over just solving linear system of equation is they may not give actual optimum point if the number of iterations is not enough to find it. They may give the optimal point that they currently obtained however if we solve linear system of equation we can get exact optimal point.

Question2 - (Implementation can be found on Appendix B)

Objective Function: $$f(x_1, x_2) = e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} + e^{-x_1-0.1}. \qquad (9.20)$$

Gradient: $$\begin{pmatrix} e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} - e^{-x_1-0.1} \\ 3e^{x_1+3x_2-0.1} - 3e^{x_1-3x_2-0.1} \end{pmatrix}$$

Hessian: $$\begin{pmatrix} e^{x_1+3x_2-0.1} + e^{x_1-3x_2-0.1} + e^{-x_1-0.1} & 3e^{x_1+3x_2-0.1} - 3e^{x_1-3x_2-0.1} \\ 3e^{x_1+3x_2-0.1} - 3e^{x_1-3x_2-0.1} & 9e^{x_1+3x_2-0.1} + 9e^{x_1-3x_2-0.1} \end{pmatrix}$$

Newton step: $$\Delta x_{\mathrm{nt}} = -\nabla^2 f(x)^{-1} \nabla f(x)$$

Newton decrement: $$\lambda(x) = \left(\nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)\right)^{1/2}$$

Newton's method:

Algorithm 9.5 *Newton's method.*

given a starting point $x \in \mathbf{dom}\, f$, tolerance $\epsilon > 0$.

repeat

1. *Compute the Newton step and decrement.*
   $\Delta x_{\mathrm{nt}} := -\nabla^2 f(x)^{-1} \nabla f(x)$; $\lambda^2 := \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)$.
2. *Stopping criterion.* **quit** if $\lambda^2/2 \leq \epsilon$.
3. *Line search.* Choose step size t by backtracking line search.
4. *Update.* $x := x + t\Delta x_{\mathrm{nt}}$.

Graphs: Error $f(x^{(k)}) - p^\star$ versus iteration k graphs, for problem 9.20 with different staring points. Backtracking line search with α=0.1 and β=0.7

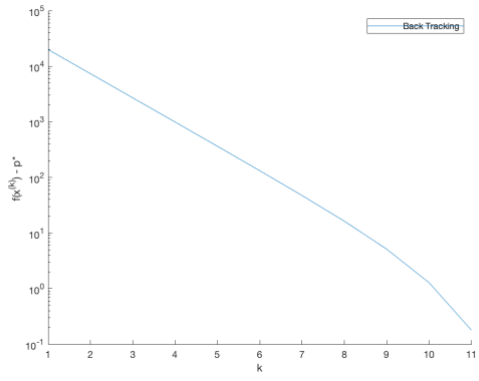


Figure-8: x=[7,1]

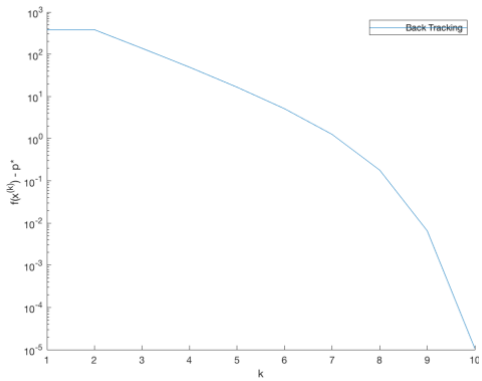


Figure-9: x=[-3,3]

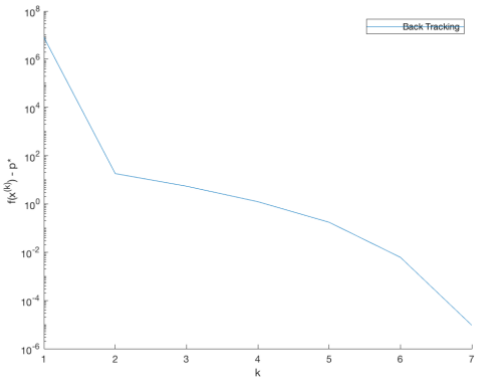


Figure-10: x=[1,5]

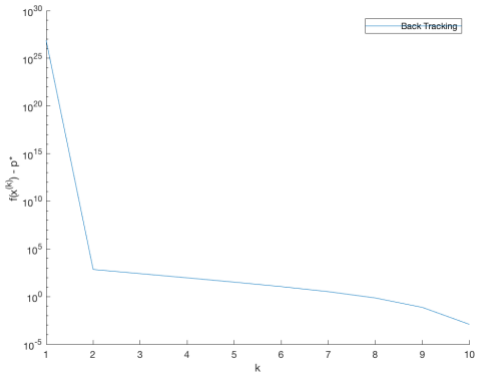


Figure-10: x=[2,20]

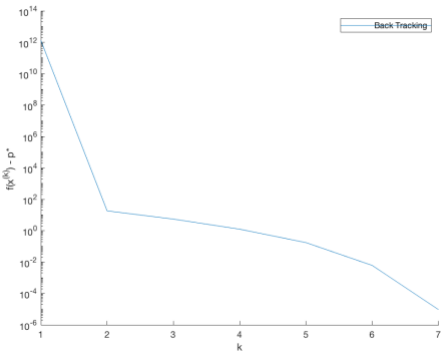


Figure-11: x=[1,9]

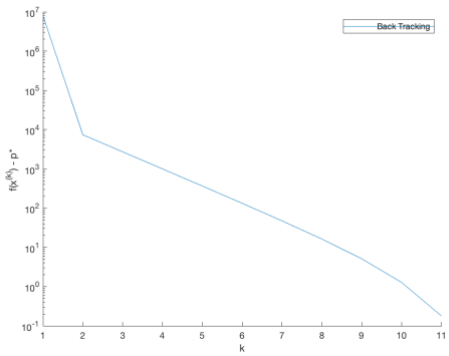


Figure-12: x=[-5,-7]

Question3 - (Implementation can be found on Appendix C)

Infeasible start Newton method:

Algorithm 10.2 *Infeasible start Newton method.*

given starting point $x \in \mathbf{dom}\, f$, ν, tolerance $\epsilon > 0$, $\alpha \in (0, 1/2)$, $\beta \in (0, 1)$.

repeat

1. Compute primal and dual Newton steps Δx_{nt}, $\Delta \nu_{\mathrm{nt}}$.
2. *Backtracking line search on* $\|r\|_2$.
   $t := 1$.
   while $\|r(x + t\Delta x_{\mathrm{nt}}, \nu + t\Delta \nu_{\mathrm{nt}})\|_2 > (1 - \alpha t)\|r(x, \nu)\|_2$, $t := \beta t$.
3. *Update.* $x := x + t\Delta x_{\mathrm{nt}}$, $\nu := \nu + t\Delta \nu_{\mathrm{nt}}$.

until $Ax = b$ and $\|r(x, \nu)\|_2 \leq \epsilon$.

Results obtained for couple of runs of algorithm are provided on Figures 13-15.

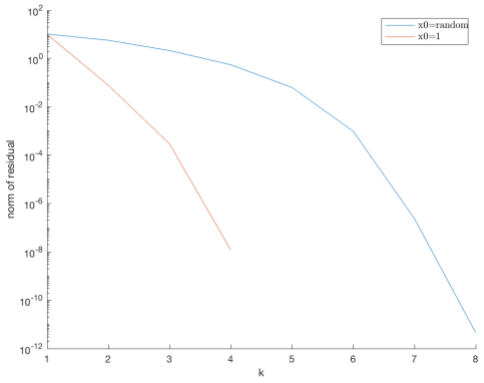


Figure-13

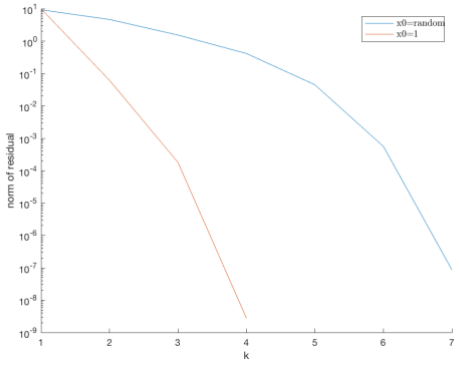


Figure-14

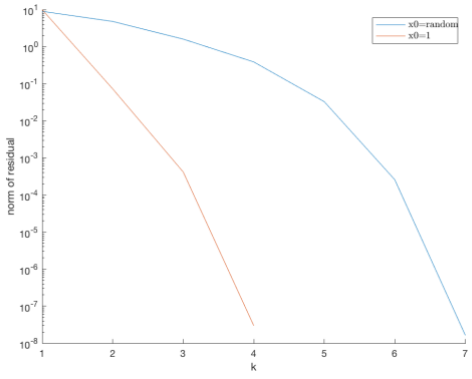


Figure-15

Solving KKT system

$$\begin{array}{ll} \text{minimize} & f(x) = \sum_{i=1}^{n} x_i \log x_i \\ \text{subject to} & Ax = b, \end{array}$$

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x_{\text{nt}} \\ w \end{bmatrix} = \begin{bmatrix} -g \\ 0 \end{bmatrix}$$

Where g = $\nabla f(x)_i = \log x_i + 1, \quad i = 1, \ldots, n.$ and $\nabla^2 f(x) = \mathbf{diag}(x)^{-1}.$

So we have: $\nabla^2 f(x)\, \Delta x_{nt} + A^T w = -g$ and $A\, \Delta x_{nt} = 0$

$\Delta x_{nt} = -(A^T w + g)\, \text{diag}(x)$ if we plug in: $A\, \Delta x_{nt} = -A\, \text{diag}(x)\, (A^T w + g) = 0$

$A\, \text{diag}(x)\, A^T w = -A\, \text{diag}(x)\, g$

Question 4 - (Implementation can be found on Appendix D)

In order to test feasibility of a problem of the form A we need to convert them to form B and solve problem B. Based on value of optimal solution we can decide if our problem is feasible or not.

Problem A =>

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b, \end{array}$$

Problem B =>

$$\begin{array}{ll} \text{minimize} & s \\ \text{subject to} & f_i(x) \leq s, \quad i = 1, \ldots, m \\ & Ax = b \end{array}$$

Assuming optimal value of Problem B is p*:

1) If p*<0 then Problem A has strictly feasible solution.
2) If p*>0 then Problem A is infeasible.
3) If p*=0 and minimum is attained at x* and s* = 0 then Problem A is feasible but not strictly feasible. If p*=0 and the minimum is not attained then Problem A is infeasible.

So algorithm is:

- Problem B = Convert LP to format B.
- P=Solve problem B
- Feasibility=Check value of P
- Return Feasibility

The choice of the parameter μ affects the number of inner and outer iteration. For small μ there is a small number of Newton steps per outer iteration so the outer iteration is large because since each outer iteration reduces the gap by only a small amount.

On the other hand, if μ is large our previous iteration is not good approximation of our current iteration so there is many inner iteration.

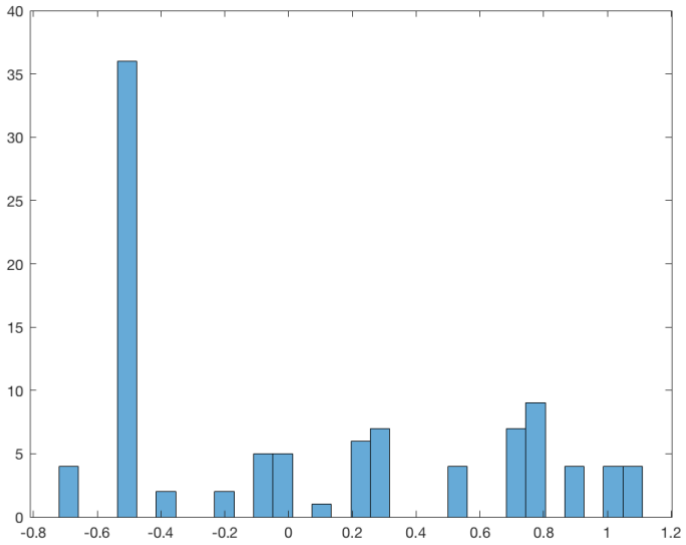


Figure-16: $b_i - a_i x_{max}$ graph

Question 5 (Implementation can be found on Appendix E)
For this problem, I was able to get F matrix and y and x vectors for
n=10;
k=5;
S=3;
But could not solve whole problem.

F^T =

```
  0.3106 + 0.0593i   0.3106 + 0.0593i   0.3106 + 0.0593i   0.3106 + 0.0593i   0.3106 + 0.0593i
  0.2940 + 0.1164i   0.2940 + 0.1164i   0.2940 + 0.1164i   0.2940 + 0.1164i   0.2940 + 0.1164i
  0.2670 + 0.1694i   0.2670 + 0.1694i   0.2670 + 0.1694i   0.2670 + 0.1694i   0.2670 + 0.1694i
  0.2305 + 0.2165i   0.2305 + 0.2165i   0.2305 + 0.2165i   0.2305 + 0.2165i   0.2305 + 0.2165i
  0.1859 + 0.2558i   0.1859 + 0.2558i   0.1859 + 0.2558i   0.1859 + 0.2558i   0.1859 + 0.2558i
  0.1346 + 0.2861i   0.1346 + 0.2861i   0.1346 + 0.2861i   0.1346 + 0.2861i   0.1346 + 0.2861i
  0.0786 + 0.3063i   0.0786 + 0.3063i   0.0786 + 0.3063i   0.0786 + 0.3063i   0.0786 + 0.3063i
  0.0199 + 0.3156i   0.0199 + 0.3156i   0.0199 + 0.3156i   0.0199 + 0.3156i   0.0199 + 0.3156i
 -0.0396 + 0.3137i  -0.0396 + 0.3137i  -0.0396 + 0.3137i  -0.0396 + 0.3137i  -0.0396 + 0.3137i
 -0.0977 + 0.3008i  -0.0977 + 0.3008i  -0.0977 + 0.3008i  -0.0977 + 0.3008i  -0.0977 + 0.3008i
```

X^T = 1 0 1 0 1 0 0 0 0 0

Y^T = 0.7635 + 0.4845i 0.7635 + 0.4845i 0.7635 + 0.4845i 0.7635 + 0.4845i 0.7635 + 0.4845i

Part 2-(Implementation can be found on Appendix F)

In this part, we will look at a small problem that I encountered on my research and solved with convex optimization.

Problem definition: We have two sets (Set1 and Set2) of nodes. Our aim is to measure RTT between each node from Set1 to each node from Set2.
Example:

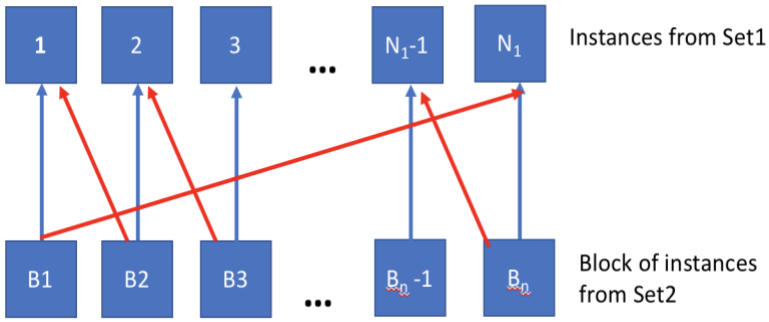
Set1={a,b,c} and Set2={1,2,3}
We need:

RTT1=1->a	RTT2=1->b	RTT3=1->c	RTT4=2->a
RTT5=2->b	RTT6=2->c	RTT7=3->a	RTT8=3->b
RTT9=3->c			

But problem is we want to do it as fast as possible, without being blocked by any node and without overwhelming nodes with pings.
Restrictions that we encounter are rate limits by platforms (RipeAtlas) that we use.

So we have 100 nodes in our Set1 and 10000 nodes in Set2.

Algorithm that we use partitions Set2 into blocks of nodes as shown on the right.
Nodes in each block pings only one instance in one iteration. In second iteration we rotate blocks and ping instances again.
Assume block size is 100 so number of blocks will match number of instances.
In 1st iteration 100 nodes in B1 will ping only instance 1 (blue line). In 2nd iteration we will rotate blocks so 100 nodes from B2 will ping instance 1 (red line).



Number of blocks will be equal to number of instances in Set1, but depending on size of block this procedure may repeat so we will have one outer loop in our algorithm. So overall algorithm is provided below.

Length1=number of instances in Set1
Length2=number of instances in Set2
Blen=number of instances in one block
OverallTime=0
For (i=0; i<Length2/(Length1*Blen); i++)
    For (j=0; j<Length1; j++) //number of rotations
        Time=time spend to ping each instance with each block
        OverallTime= OverallTime +Time
As you may assume our aim is to minimize OverallTime.

minimize OverallTime
where OverallTime= Length2/Blen * Time (given Blen)

Now let me discuss how Time is calculated. Which is time needed to achieve operation on the right. This operation is achieved by RipeAtlas by specifying some parameters.

Assume **1** is **instance1,** and we have 60 nodes in B1. If we let to ping instance1 with 60 nodes at once, we will not be able to get actual value of each RTT because they will overwhelm instance1 so RTTs will increase.

So we need to specify parameters like **interval** and **spread** to RipeAtlas.
So assume parameters are:
Source = 60 probes (nodes in B1)
Target = 1 instance (node 1)
Interval = 240 (seconds)
Spread = 240 (seconds)

So in each 4second, target will be pinged by 1 node from Sources.
4second comes from (240/60) basically, Ripe spreads 60 pings to my interval evenly.
From now on we will use terminology SpreadTime as time between two RTT measurements.
SpreadTime is 4, for case above.

This operation takes overall: 2*interval + initializationTime
Where initializationTime is time for RipeAtlas to allocate nodes (which is usually 240 seconds).

So variables:

Length1:	number of instances in Set1
Length2:	number of instances in Set2
Blen:	number of nodes in one block
Interval:	should be given as parameter to RipeAtlas
initializationTime:	240 seconds (Usually).
SpreadTime:	Interval/Blen
Time:	2*interval + initializationTime
OverallTime:	(Length2/Blen)* Time

Our main goal:

minimize **f_0(Blen, Interval) = OverallTime**

subject to **1 <= Blen <= 1000 //Because of Ripe Atlas rate limit**
SpreadTime >=5seconds //we don't want overwhelm instance

So, I implemented it and provided the code on Appendix F.
So here is result that I obtained.

```
>> part2

Calling SDPT3 4.0: 8 variables, 3 equality constraints
------------------------------------------------------------

 num. of constraints =  3
 dim. of sdp    var  =  2,   num. of sdp  blk  =  1
 dim. of linear var  =  3
 dim. of free   var  =  2 *** convert ublk to lblk
*****************************************************************
   SDPT3: Infeasible path-following algorithms
*****************************************************************
 version  predcorr  gam  expon  scale_data
   HKM      1      0.000   1        0
it pstep dstep pinfeas dinfeas  gap      prim-obj      dual-obj    cputime
```

```
-----------------------------------------------------------------
 0|0.000|0.000|6.0e+00|1.2e+01|1.1e+05| 1.001000e+03  0.000000e+00| 0:0:00| chol  1  1
 1|0.886|1.000|6.9e-01|5.0e-01|6.8e+03| 8.465515e+02 -9.928882e+02| 0:0:00| chol  1  1
 2|0.754|0.980|1.7e-01|1.6e-01|6.4e+02| 2.380761e+02  1.006438e+03| 0:0:00| chol  1  1
 3|0.104|1.000|1.5e-01|4.5e-02|3.0e+03| 2.200873e+02  3.453476e+04| 0:0:00| chol  1  1
 4|0.055|1.000|1.4e-01|1.3e-02|1.1e+05| 2.260185e+02  7.522898e+06| 0:0:00| chol  1  1
 5|0.000|0.000|1.4e-01|1.5e-02|2.2e+05| 1.809516e+03  1.620960e+07| 0:0:00| chol  1  1
 6|0.000|0.000|1.4e-01|1.8e-02|3.3e+05| 1.140094e+04  2.394977e+07| 0:0:00| chol  1  1
 7|0.000|0.000|1.4e-01|2.2e-02|8.9e+05| 1.407240e+03  6.339256e+07| 0:0:00| chol  1  1
 8|0.000|0.012|1.4e-01|2.7e-02|8.7e+06| 2.975366e+05  4.132196e+08| 0:0:00| chol  1  1
 9|0.002|0.001|1.4e-01|3.4e-02|2.9e+07| 1.388934e+07  6.554891e+08| 0:0:00| chol  1  1
10|0.002|0.001|1.4e-01|4.4e-02|3.0e+07| 1.396356e+06  1.050426e+09| 0:0:00| chol  1  1
11|0.013|0.000|1.4e-01|5.7e-02|1.6e+09| 1.624872e+09  1.386224e+08| 0:0:00| chol  1  1
12|0.040|0.597|1.3e-01|2.3e-02|1.9e+09| 1.585673e+09  4.471905e+09| 0:0:00| chol  1  1
13|0.356|0.656|8.7e-02|7.8e-03|1.9e+09| 3.133754e+09  4.902960e+09| 0:0:00| chol  1  1
14|1.000|0.369|2.9e-12|2.1e-02|1.1e+10| 1.462169e+10  3.691352e+09| 0:0:00| chol  1  1
15|0.750|0.934|1.3e-12|1.4e-03|5.0e+09| 1.091863e+10  5.961838e+09| 0:0:00| chol  1  1
16|1.000|0.900|4.6e-14|1.4e-04|9.3e+08| 1.009062e+10  9.171585e+09| 0:0:00| chol  1  1
17|0.973|0.983|4.0e-15|2.7e-06|2.1e+07| 9.610575e+09  9.590044e+09| 0:0:00| chol  1  1
18|0.980|0.982|1.8e-15|2.2e-07|3.9e+05| 9.600215e+09  9.599824e+09| 0:0:00| chol  1  1
19|0.955|0.979|9.0e-16|3.0e-08|1.4e+04| 9.600010e+09  9.599996e+09| 0:0:00| chol  1  1
20|0.950|0.968|9.8e-16|1.3e-10|1.0e+03| 9.600001e+09  9.600000e+09| 0:0:00| chol  1  1
 number of iterations   = 23
 primal objective value =  9.60000000e+09
 dual   objective value =  9.60000000e+09
 gap := trace(XZ)       = 4.63e-01
 relative gap           = 2.41e-11
 actual relative gap    = 2.19e-11
 rel. primal infeas (scaled problem)   = 7.83e-16
 rel. dual     "        "       "      = 2.73e-12
 rel. primal infeas (unscaled problem) = 0.00e+00
 rel. dual     "        "       "      = 0.00e+00
 norm(X), norm(y), norm(Z) = 9.6e+09, 8.0e+07, 4.1e+08
 norm(A), norm(b), norm(C) = 6.5e+00, 1.0e+03, 2.0e+00
 Total CPU time (secs)  = 0.49
 CPU time per iteration = 0.02
 termination code       =  0
 DIMACS: 7.9e-16  0.0e+00  2.7e-12  0.0e+00  2.2e-11  2.4e-11
-----------------------------------------------------------------

------------------------------------------------------------
Status: Solved
Optimal value (cvx_optval): +9.6e+09


blen =

   23.9999


interval =

  119.9993
```

So block size should be 24 and interval is 120seconds. In each 5 second one node will ping instance. One iteration will finish in

2 * interval + initializationTime =2*120+240=420(seconds)

Appendix A: (Question 1)

```
function optimal = mygradient(varargin)

Q1= randi(2*52,10,10);

Q=Q1*Q1';

d=eig(Q);
emax=max(d);
emin=min(d);
condition_number=emax/emin;
condition_number

q=randi(2*52,10,1);

%q=[34;6;10;29;50];

x_starting_point=randi(2*52,10,1);
%x_starting_point=[76;9;9;14;12];

%maximum number of iteration
maximumNumberOfIteration=200000;
%tol = 1e-6;                    % termination tolerance
tol=1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%f = @(x) 0.5*x'*Q*x + q'*x      %objective function

%starting point, will also be used as current optimal point
```

```
x_sP =  x_starting_point;
ls1=[];
ls2=[];

%calling gradient descent
%0=gradient descent
[optimalWithBackTrackLinesearch,ls1]=gradient_method(x_sP, 1, maximumNumberOfIteration,Q,q,tol,0);
[optimalWithExactLinesearch,ls2]=gradient_method(x_sP, 0, maximumNumberOfIteration,Q,q,tol,0);


%optimalWithBackTrackLinesearch
%optimalWith
opB=objectiveF(optimalWithBackTrackLinesearch, Q, q);
opE=objectiveF(optimalWithExactLinesearch, Q, q);
opB
opE


data{1}=ls1;
data{2}=ls2;

hold on
xlabel('k'); % x-axis label
ylabel('f(x^(^k^)) - p* '); % y-axis label
set(gca, 'YScale', 'log');
cellfun(@(x) plot(x), data);
```

```
[hleg1, hobj1]=legend('BackTrack','Exact');
textobj = findobj(hobj1, 'type', 'text');
set(textobj, 'Interpreter', 'latex', 'fontsize', 10);

%calling steepest descent
%1=steepest descent
[optimalWithBackTrackLinesearch,ls1]=gradient_method(x_sP, 1, maximumNumberOfIteration,Q,q,tol,1);
[optimalWithExactLinesearch,ls2]=gradient_method(x_sP, 0, maximumNumberOfIteration,Q,q,tol,1);

opB=objectiveF(optimalWithBackTrackLinesearch, Q, q);
opE=objectiveF(optimalWithExactLinesearch, Q, q);
opB
opE
```

```
function [optimal_value,ls] = gradient_method(x_initial, which_linesearch, maximumNumberOfIteration,Q,q,tol,which_method)
x_sP=x_initial;
ls=[];
P = diag(diag(Q));
% initialize gradient norm, iteration counter
gnorm = inf;  numberOfiteration = 0;
ls(end+1) = objectiveF(x_sP,Q,q);
while and(gnorm>=tol, numberOfiteration <= maximumNumberOfIteration)
    gradient=gradienter(x_sP,Q,q);            %calculate gradient
    if which_method == 0 %gradient
        descentDirection=-gradient;           %determine descent direction
    end
    if which_method == 1 %steepest
        descentDirection=-inv(P)*gradient;
    end

    %stepSize=0.1;                                %determine stepsize
    if which_linesearch == 1
        stepSize=BackTrackLinesearch(Q,q,descentDirection,x_sP,0.5,0.1); %B alpha
    end
    if which_linesearch == 0
        stepSize=ExactLinesearch(Q,q,descentDirection,x_sP);
    end
    x_sP =  x_sP + (stepSize * descentDirection);  %update Current optimal point

    ls(end+1) = objectiveF(x_sP,Q,q);
    %update stopping criterion
    gnorm = norm(gradient);
    numberOfiteration = numberOfiteration + 1;
end
numberOfiteration
pStar = objectiveF(x_sP,Q,q);
for k=1:length(ls)
    curw=ls(k);
    ls(k)=curw-pStar;
end
```

```
optimal_value=x_sP;

function t = ExactLinesearch(Q,q,descentDirection,x)
t=1;
gradient =  gradienter(x,Q,q);  %vector direction
t=(gradient'*gradient)/(gradient'*Q*gradient);


function t = BackTrackLinesearch(Q,q,descentDirection,x,B,alpha)
t = 1;                              %stepsize
fright = objectiveF(x,Q,q);     %value
gradient =  gradienter(x,Q,q);  %vector direction
xx = x;
x = xx + t*descentDirection;    %vector -> argument for left side of while
fleft = objectiveF(x,Q,q);      %value  -> left side of while

while fleft > fright + alpha*t*(gradient'*descentDirection)
    t = t*B;
    x = xx + t*descentDirection;    %argument for left side of while
    fleft = objectiveF(x,Q,q);      %left side of while
end

function f = objectiveF(x,Q,q) %objective function
f=0.5*x'*Q*x + q'*x;             %returns value
```

```
  % define the gradient of the objective
  %returns vector (direction)
  function g = gradienter(x,Q,q)
    g = Q*x+q;
```

Appendix B: (Question 2)

```
function optimal = Q2(varargin) %Newton's Method for Unconstrained Problems

x = randi(20,2,1);
maximumNumberOfIteration=10;     %maximum number of iteration
tol = 1e-6;                      % termination tolerance
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ls=[];

%calling Newton's Method
[xoptimal,ls]=newton_method(x,maximumNumberOfIteration,tol);
optimal_Value=objectiveF(xoptimal(1),xoptimal(2));

xoptimal
optimal_Value

data{1}=ls;
hold on
xlabel('k'); % x-axis label
ylabel('f(x^(^k^)) - p* '); % y-axis label

set(gca, 'YScale', 'log');
cellfun(@(x) plot(x), data);
legend('Back Tracking');
```

```
function [xoptimalP,ls] = newton_method(x_initial,maximumNumberOfIteration,tol)

x=x_initial; ls=[];
% initialize gradient norm, iteration counter
numberOfiteration = 0;
ls(end+1) = objectiveF(x(1),x(2));
while ( numberOfiteration <= maximumNumberOfIteration)

    gradient=gradienter(x(1),x(2));             %calculate gradient
    hessian=hessianer(x(1),x(2));               %calculate hessian

    newton_step=-inv(hessian)*gradient;     %calculate newton_step

    newton_decrement_sq=gradient'*inv(hessian)*gradient;

    if newton_decrement_sq/2 <= tol
        break
    end

    stepSize=BackTrackLinesearch(x,newton_step,0.7,0.1);  %B, Alpha
    x =  x + (stepSize * newton_step);

    ls(end+1) = objectiveF(x(1),x(2));
    numberOfiteration = numberOfiteration + 1;   %update stopping criterion
end
```

```
pStar = objectiveF(x(1),x(2));
for k=1:length(ls)
    curw=ls(k);
    ls(k)=curw-pStar;
end
xoptimalP=x;


function t = BackTrackLinesearch(x,descentDirection,B,alpha)
t = 1;
fright = objectiveF(x(1),x(2));
gradient =  gradienter(x(1),x(2));
xx = x;
x = xx + t*descentDirection;
fleft = objectiveF(x(1),x(2));

while fleft > fright + alpha*t*(gradient'*descentDirection)
    t = t*B;
    x = xx + t*descentDirection;
    fleft = objectiveF(x(1),x(2));
end
function f = objectiveF(x1,x2)
f=exp(x1+3*x2-0.1)+exp(x1-3*x2-0.1)+exp(-x1-0.1);
```

```
function g = gradienter(x1,x2)
r00=exp(x1+3*x2-0.1)+exp(x1-3*x2-0.1)-exp(-x1-0.1);
r10=3*exp(x1+3*x2-0.1)-3*exp(x1-3*x2-0.1);
g=[r00;r10];

function h = hessianer(x1,x2)
r00=exp(x1+3*x2-0.1)+exp(x1-3*x2-0.1)+exp(-x1-0.1);
r01=3*exp(x1+3*x2-0.1)-3*exp(x1-3*x2-0.1);
r10=3*exp(x1+3*x2-0.1)-3*exp(x1-3*x2-0.1);
r11=9*exp(x1+3*x2-0.1)+9*exp(x1-3*x2-0.1);

h = [r00 r01;r10 r11];
```

Appendix C: (Question 3)

```
MAXITERS = 100; %maximum iteration
ALPHA = 0.01;
BETA = 0.5;
RESTOL = 1e-7;  %tolerance
p=30; %30
n=100; %100
x1=rand(1,n,1);
x1=x1';
x2=randi(1,n,1);


A=randi(100,p,n); %p by n matrix

while (rank(A,1e-16) ~= p)
A=randi(100,p,n);
end

res1=Computation(A,x1,MAXITERS,ALPHA,BETA,RESTOL,p,n);
res2=Computation(A,x2,MAXITERS,ALPHA,BETA,RESTOL,p,n);

data{1}=res1;
data{2}=res2;
hold on
set(gca, 'YScale', 'log');
xlabel('k'); % x-axis label
ylabel('norm of residual'); % y-axis label
cellfun(@(x) plot(x), data);


[hleg1, hobj1]=legend('x0=random','x0=1');
textobj = findobj(hobj1, 'type', 'text');
set(textobj, 'Interpreter', 'latex', 'fontsize', 10);
function res = Computation(A,x,MAXITERS,ALPHA,BETA,RESTOL,p,n)
    v=zeros(p,1);
    b=A*x; %
    res=[];
    for i=1:MAXITERS
       gradient=1+log(x);
       r = [gradient+A'*v;  A*x-b];
       res = [res, norm(r)]; %add elements to end
       sol = -[diag(1./x) A';  A zeros(p,p)] \ r;
       Dx = sol(1:n);
       Dnu = sol(n+[1:p]);
       if (norm(r) < RESTOL)
           break;
       end;
       t=1;
       while (min(x+t*Dx) <= 0)
           t = BETA*t;
       end;
       while norm([1+log(x+t*Dx)+A'*(v+Dnu);  A*(x+Dx)-b]) > (1-ALPHA*t)*norm(r)
           t=BETA*t;
       end;
       x=x+t*Dx;
       v=v+t*Dnu;
    end;
end
```

Appendix D: (Question 4)

```
function x = Q4()

H = diag(randi([0 100],1,100));
q=randi([1 5],1,100)';

x0=randi([0 101],1,100)';

for i = 1:50
    P{i} = diag(randi([1 5],1,100));
    r{i} = randi([-5 5],1,100)';
end
b = randi([-5 5],1,50)';
x = myfun(x0,H, q, P, r, b,1e-16,100);
nbins = 30;
h = histogram(x,nbins)
end

function f=con(x, P, r, b)
    f = zeros(length(b),1);
    for i = 1:length(b)
      f(i) = x'*P{i}*x/2 + r{i}'*x - b(i);
    end
end

function  [J, W]=jfun(x,z, P, r, b)
    J = zeros(length(b),length(x));
    W = zeros(length(x));
    for i = 1:length(b);
      J(i,:) = (P{i}*x + r{i})';
      W = W + z(i)*P{i};
    end
end
```

```
function x = myfun (x,H, q, P, r, b, tolerance, miter)
  amax = 0.99;     eps = 1e-8;   beta = 0.7;    mmin= 1e-16;
  amin = 1e-12;    emax= 0.25;   teta = 0.01;   smax = 0.5;

  c  = con(x, P, r, b);  alpha = 0;
  nm = length(x) + length(c);    z = ones(length(c),1);
  for iter = 1:miter

    f = x'*H*x/2 + q'*x;
    c = con(x, P, r, b);
    [J W] = jfun(x,z, P, r, b);
    g =  H*x + q;
    B=H;
    rmat = [g + J'*z;c.*z];
    eta        = min(emax,norm(rmat)/nm);
    sigma      = min(smax,sqrt(norm(rmat)/nm));
    dualitygap = -c'*z;
    mu         = max(mmin,sigma*dualitygap/length(c));

    if norm(rmat)/nm < tolerance
      break
    end
    S  = diag(sparse(z./(c-eps)));
    gb = g - mu*J'*(1./(c-eps));
    px = (B + W - J'*S*J) \ (-gb);
    pz = -(z + mu./(c-eps) + S*J*px);
    alpha = amax;
    is    = find(z + pz < 0);
    if length(is)
      alpha = amax * min(1,min(z(is) ./ -pz(is)));
    end
```

```
    pi  = f - c'*z - mu*sum(log(c.^2.*z+eps));
    dpi = px'*(g - J'*z - 2*mu*J'*(1./(c-eps))) - pz'*(c + mu./(z+eps));

    while true
      xnew   = x + alpha * px;
      znew   = z + alpha * pz;

      c      = con(xnew, P, r, b);
      pnew = (xnew'*H*xnew/2 + q'*xnew) - c'*z - mu*sum(log(c.^2.*z+eps));

      if sum(c > 0) == 0 & pnew < pi + teta*eta*alpha*dpi
        x = xnew;   z = znew;
        break
      end

      alpha = alpha * beta;
      if alpha < amin
           fprintf("small stepsize\n");
           break
      end
    end
  end
end
```

Appendix E: (Question 5)

```
n=10;
k=5;

S=3;
x = [zeros(1,n-S), ones(1,S)];
x = x(randperm(n1));
freq=S/n;

nvec = 0:1:n-1; % row vector for n
kvec = 0:1:k-1; % row vecor for k
F = kvec'*nvec; % creates a k by n matrix

for r = 1:k;
   for c=1:n;
      F(r,c) =(1/(sqrt(N))) * exp(-1j*2*pi*c*freq/N);
   end;
end;

y = (F*x');
F'
x
y'
```

Appendix F: (Part2)

```
   cvx_begin
      variables blen(1) interval(1) length1(1) length2(1) initializationTime(1) time(1) overallTime(1)

      minimize(quad_over_lin(time,blen))
      subject to

           time == (2*interval + initializationTime)*1000

           1 <= blen
           blen <= 1000
           initializationTime == 240
           5*blen-interval<=0
           length2 == 1000
   cvx_end
   blen
   interval
```