Technical University of Crete
School of Electrical and Computer Engineering
Course: **Convex Optimization**
Exercise 2 (110/500)
Report Delivery Date: 2 November 2021
Instructor: Athanasios P. Liavas

Student: Alevrakis Dimitrios 2017030001

A. We consider a simple quadratic optimization problem. Let $f : \mathbb{R}^n \to \mathbb{R}$. For fixed $\mathbf{x} \in \mathbb{R}$ and $m > 0$, let $g_{\mathbf{x}} : \mathbb{R}^n \to \mathbb{R}$ be defined as

$$g_{\mathbf{x}}(\mathbf{y}) \coloneqq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{m}{2}||\mathbf{y} - \mathbf{x}||_2^2 \tag{1}$$

i. We will be calculating $\nabla g_{\mathbf{x}}(\mathbf{y})$:
First we will bring $g_{\mathbf{x}}(\mathbf{y})$ in the form:
$g_{\mathbf{x}}(\mathbf{y}) = \frac{1}{2}\mathbf{y}^T P\mathbf{y} + q^T\mathbf{y} + r$, where $P \in \mathbb{R}^{nxn},\ P = P^T,\ q \in \mathbb{R}^n$ and $r \in \mathbb{R}$

$$\begin{aligned} g_{\mathbf{x}}(\mathbf{y}) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{m}{2}||\mathbf{y} - \mathbf{x}||_2^2 \\ &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T\mathbf{y} - \nabla f(\mathbf{x})^T\mathbf{x} + \frac{m}{2}(\mathbf{y} - \mathbf{x})^T(\mathbf{y} - \mathbf{x}) \\ &= f(\mathbf{x}) - \nabla f(\mathbf{x})^T\mathbf{x} + \nabla f(\mathbf{x})^T\mathbf{y} + \frac{m}{2}(\mathbf{y}^T - \mathbf{x}^T)(\mathbf{y} - \mathbf{x}) \\ &= f(\mathbf{x}) - \nabla f(\mathbf{x})^T\mathbf{x} + \nabla f(\mathbf{x})^T\mathbf{y} + \frac{m}{2}(\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{x} - \mathbf{x}^T\mathbf{y} + \mathbf{x}^T\mathbf{x}) \\ &\overset{\mathbf{y}^T\mathbf{x}=\mathbf{x}^T\mathbf{y}}{=} f(\mathbf{x}) - \nabla f(\mathbf{x})^T\mathbf{x} + \nabla f(\mathbf{x})^T\mathbf{y} + \frac{m}{2}(\mathbf{y}^T\mathbf{y} - 2\mathbf{x}^T\mathbf{y} + \mathbf{x}^T\mathbf{x}) \\ &= f(\mathbf{x}) - \nabla f(\mathbf{x})^T\mathbf{x} + \frac{m}{2}\mathbf{x}^T\mathbf{x} + (\nabla f(\mathbf{x}) - m\mathbf{x})^T\mathbf{y} + \frac{1}{2}\mathbf{y}^T m\mathbb{I}_n\mathbf{y} \end{aligned} \tag{2}$$

Where $\mathbb{I}_n \in \mathbb{R}_{nxn}$ the identity matrix. For $\mathbb{I}_n$ is also true that $\mathbb{I}_n = \mathbb{I}_n^T$
Therefore we have:

$$\begin{aligned} P &= m\mathbb{I}_n \\ q &= \nabla f(\mathbf{x}) - m\mathbf{x} \\ r &= f(\mathbf{x}) - \nabla f(\mathbf{x})^T\mathbf{x} + \frac{m}{2}\mathbf{x}^T\mathbf{x} \end{aligned} \tag{3}$$

and $\nabla g_{\mathbf{x}}(\mathbf{y}) = P\mathbf{y} + q = m\mathbb{I}_n y + \nabla f(\mathbf{x}) - m\mathbf{x} = my + \nabla f(\mathbf{x}) - m\mathbf{x}$

ii. In order to find the optimal point and value we will prove that $g_{\mathbf{x}}\mathbf{y}$ is strictly convex and then set $\nabla g_{\mathbf{x}}\mathbf{y} = 0$.

- Since $g_{\mathbf{x}}(\mathbf{y})$ is quadratic, it is sufficient to prove that P is definite positive. By definition a matrix $M \in \mathbb{R}^{nxn}$ is definite positive if $\mathbf{z}^T M\mathbf{z} > 0 \forall z \in \mathbb{R}^n - 0$.

$$\mathbf{z}^T P\mathbf{z} = \mathbf{z}^T m\mathbb{I}_n\mathbf{z} = m\mathbf{z}^T\mathbf{z} = m||z||_2^2 > 0 \forall z \in \mathbb{R}^n - 0, m > 0 \tag{4}$$

  Therefore $g_{\mathbf{x}}(\mathbf{y})$ is strictly convex.

- The optimal point $y_* = argmin_y g_{\mathbf{x}}(\mathbf{y})$:

$$\begin{aligned} \nabla g_{\mathbf{x}}(\mathbf{y}_*) = 0 \iff \\ my_* + \nabla f(\mathbf{x}) - m\mathbf{x} = 0 \iff \\ y_* = \mathbf{x} - \frac{1}{m}\nabla f(\mathbf{x}) \end{aligned} \tag{5}$$

  We replace y_* in $g_{\mathbf{x}}(\mathbf{y})$ to find the optimal value.

$$\begin{aligned} g_{\mathbf{x}}(\mathbf{y}_*) &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}_* - \mathbf{x}) + \frac{m}{2}||\mathbf{y}_* - \mathbf{x}||_2^2 \\ &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{x} - \frac{1}{m}\nabla f(\mathbf{x}) - \mathbf{x}) + \frac{m}{2}||\mathbf{x} - \frac{1}{m}\nabla f(\mathbf{x}) - \mathbf{x}||_2^2 \\ &= f(\mathbf{x}) + \nabla f(\mathbf{x})^T(-\frac{1}{m}\nabla f(\mathbf{x})) + \frac{m}{2}|| - \frac{1}{m}\nabla f(\mathbf{x})||_2^2 \\ &= f(\mathbf{x}) - \frac{1}{m}||\nabla f(\mathbf{x})||_2^2 + \frac{1}{2m}||\nabla f(\mathbf{x})||_2^2 \\ &= f(\mathbf{x}) - \frac{1}{2m}||\nabla f(\mathbf{x})||_2^2 \end{aligned} \tag{6}$$

B. The problem:

$$\underset{minimize}{x \in \mathbb{R}^n} f(x) = \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x} \tag{7}$$

Where $\mathbf{P} \in \mathbb{R}^n, \mathbf{P} = \mathbf{P}^T \succ \mathbf{0}$ and $\mathbf{q} \in \mathbb{R}^{\mathbf{n}}$

i. We construct a random positive definite matrix $\mathbf{P}$ using the the fact that it can be expressed as $\mathbf{P} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ since it is positive definite.
Where $\mathbf{U}, \mathbf{\Lambda} \in \mathbb{R}^{nxn}, \mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}_n$ and $\mathbf{\Lambda} = diag(\lambda_1, ..., \lambda_n)$ with $\lambda_i > 0, i \in [1, n]$. The columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{P}$ and the elements of the diagonal of Λ are the eigenvalues of $\mathbf{P}$.

(a) We construct a random orthonormal $\mathbf{U}$ using the suggested method.

```
A = randn(n);
[U,~,~] = svd(A);

disp(U*U'); %Almost equal to I
disp(U'*U);
```

We observe that the matrices $\mathbf{U}\mathbf{U}^T, \mathbf{U}^T\mathbf{U}$ are almost equal to the identity matrices and thus can be considered orthonormal.

(b) We also construct the matrix Λ using the suggested method by choosing $l_{min} = 1$ and scaling l_{max} in order to control the value of the condition number.

```
%B.i.b
l_max = [1,2,10, 100, 1000];
l_min = 1;
fprintf('************************  n=%d  ************************
for i=1:size(l_max,2)
    z = l_min + (l_max(i) - l_min) * rand(n - 2, 1);
    eig_v = [l_min ; l_max(i) ; z];
    L = diag(eig_v);

    %B.ii
    CN = l_max(i)/l_min;
```

ii. As above we construct the matrices $\mathbf{U}$ and Λ in order to form the matrix $\mathbf{P}$. A random vector $\mathbf{q}$ is also made.

```
%% B.i
n=2;

%B.i.a
A = randn(n);
[U,~,~] = svd(A);
```

```
disp(U*U'); %Almost equal to I
disp(U'*U);

%B.i.b
l_max = [1,2,10, 100, 1000];
l_min = 1;
fprintf('************************  n=%d  ************************\n
for i=1:size(l_max,2)
    z = l_min + (l_max(i) - l_min) * rand(n - 2, 1);
    eig_v = [l_min ; l_max(i) ; z];
    L = diag(eig_v);

    %B.ii
    CN = l_max(i)/l_min;
    fprintf('----------------------------------------------------------
    fprintf('Condition Number = %.0f\n',CN);

    P = U*L*U';
%     chol(P); %Check that P is positive definite
    q = randn(n,1);
    f = @(x) 1/2*x'*P*x + q'*x;
```

iii. Since $\mathbf{P}$ is positive definite, f is strictly convex and the minimization problem can be solved by solving the equation $\nabla f(\mathbf{x}^*) = \mathbf{0}$

$$\nabla f(\mathbf{x}^*) = \mathbf{0} \iff \mathbf{P}\mathbf{x}^* + \mathbf{q} = 0 \iff \mathbf{x}^* = -\mathbf{P}^{-1}\mathbf{q} \tag{8}$$

```
    %B.iii
    grad_f = @(x) P*x + q;
    %Closed-Form method
    x_star_cf = -P\q;
    p_star_cf = f(x_star_cf);
%     grad_f(x_star_cf) %Check that grad_f(x_star_cf) is indeed equal
```

```
fprintf('Closed-Form Solution: p* = %f', p_star_cf);
fprintf('\n\n');
```

iv. We will now try to solve the problem using the gradient algorithm with exact end backtracking line search. The convergence analysis shows that both algorithms will have found a solution by a margin of error $\epsilon > 0$ in $k \leq \frac{log\left(\frac{f\mathbf{x_0}-P^*}{\epsilon}\right)}{log(\frac{1}{c})}$ iterations, where x_0 the initial point where we begin the search from, and $c := 1 - \frac{m}{M}$,and $c := 1 - min\{2m\alpha, \frac{2\beta\alpha m}{M}\}$ for the exact and backtracking line search algorithms respectively. (m, M the minimum and maximum eigenvalues of $\mathbf{P}$)

- When using the exact line search algorithm we consider the condition number equal to $log(\frac{1}{c})$.
  We observe that for $\mathbb{K} = 1$ the algorithm needs exactly one iteration to find the minimum. This happens because the contours form a circle and from every point the gradient "points" towards the minima.
- When using the backtracking algorithm we also observe that the algorithm converges after one iteration for any value of a and b.

v. We construct the contour plots, with levels the values $f(\mathbf{x}_k)$ with the trajectories of $\{\mathbf{x}_k\}$, for each algorithm. In the case of the backtracking algorithm we calculate the fastest and slowest solution for a pair of the parameters a and b. In reality every a and b for the backtracking algorithm converges in one

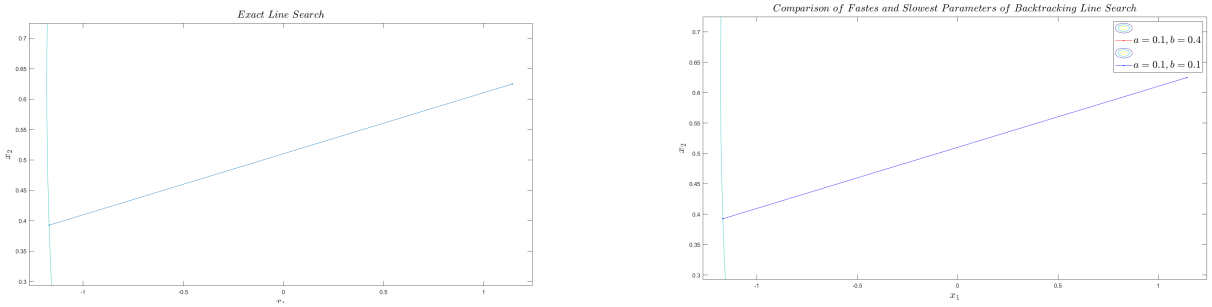


Figure 1: $\mathbb{K} = 1$

iteration for $\mathbb{K} = 1$

As $\mathbb{K}$ increases we begin to observe the zig-zag effect.

We also observe that the correct choice of the parameters a and b is important in order for the backtracking algorithm to converge fast.

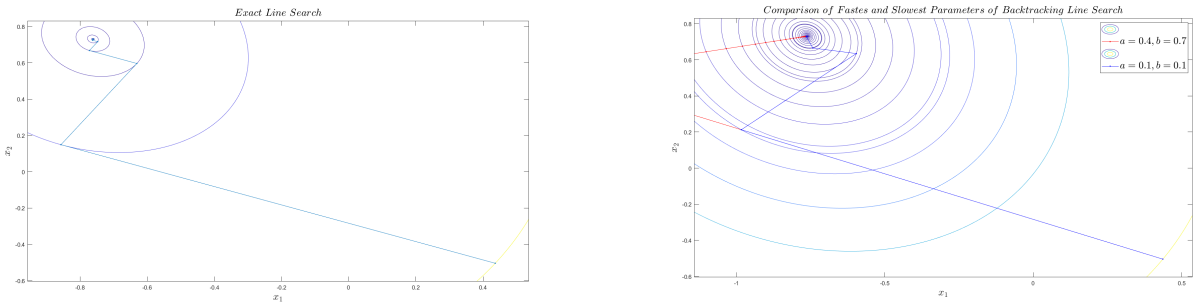


Figure 2: $\mathbb{K} = 2$

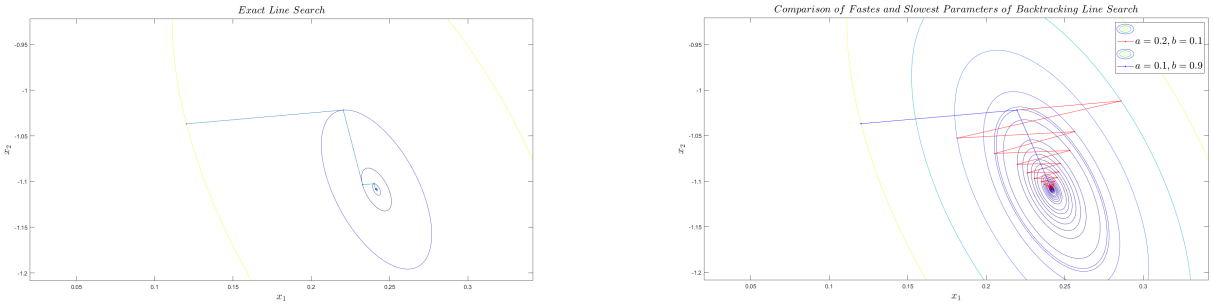


Figure 3: $\mathbb{K} = 10$

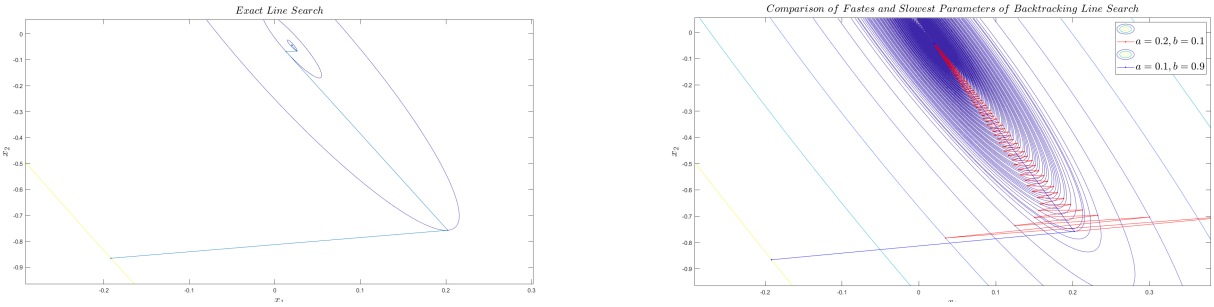


Figure 4: $\mathbb{K} = 10^2$

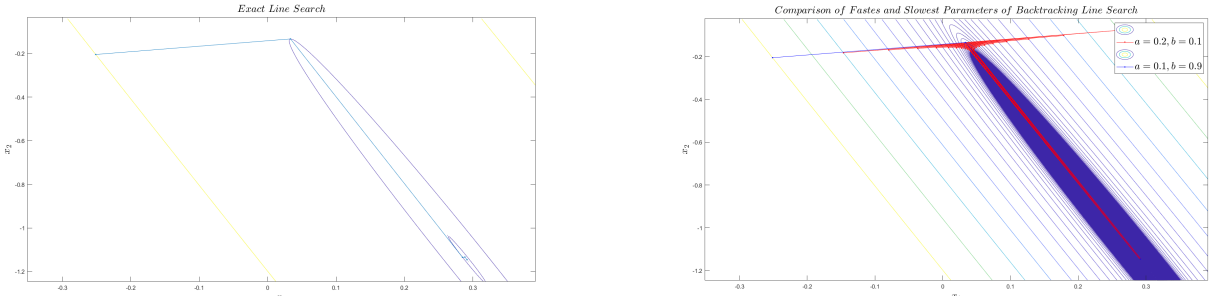


Figure 5: $\mathbb{K} = 10^3$

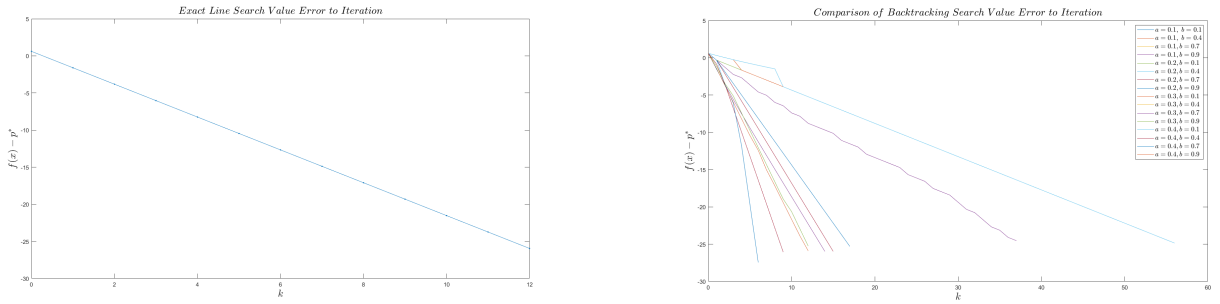


Figure 6: $\mathbb{K} = 2$

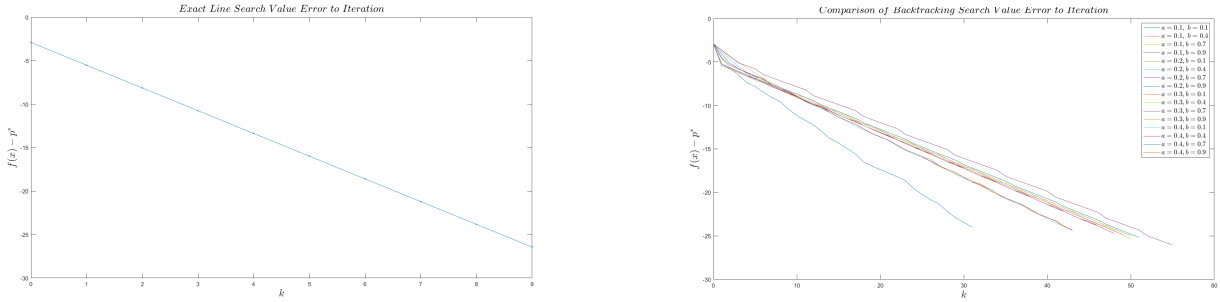


Figure 7: $\mathbb{K} = 10$

vi. For $\mathbb{K} = 1$ the plots are empty since in one iteration the algorithms find the optimal value and hence the plots are omitted.

We also observe that the slopes of each plot remain relatively the same meaning the distance from the optimal value in each iteration decreases by almost the same amount.

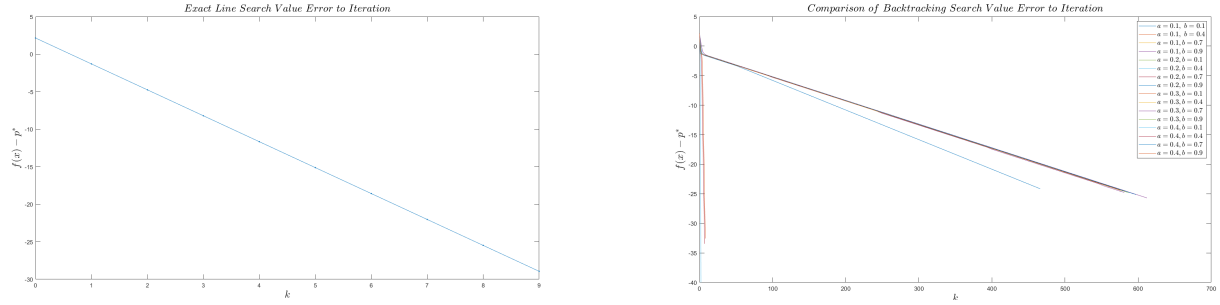


Figure 8: $\mathbb{K} = 10^2$

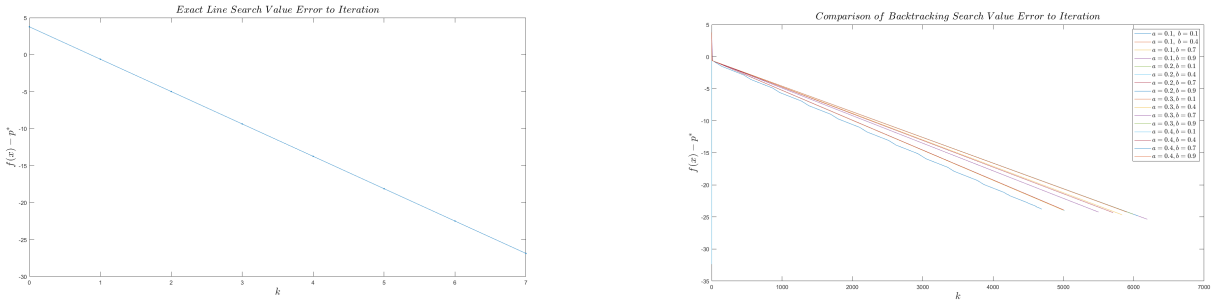


Figure 9: $\mathbb{K} = 10^3$

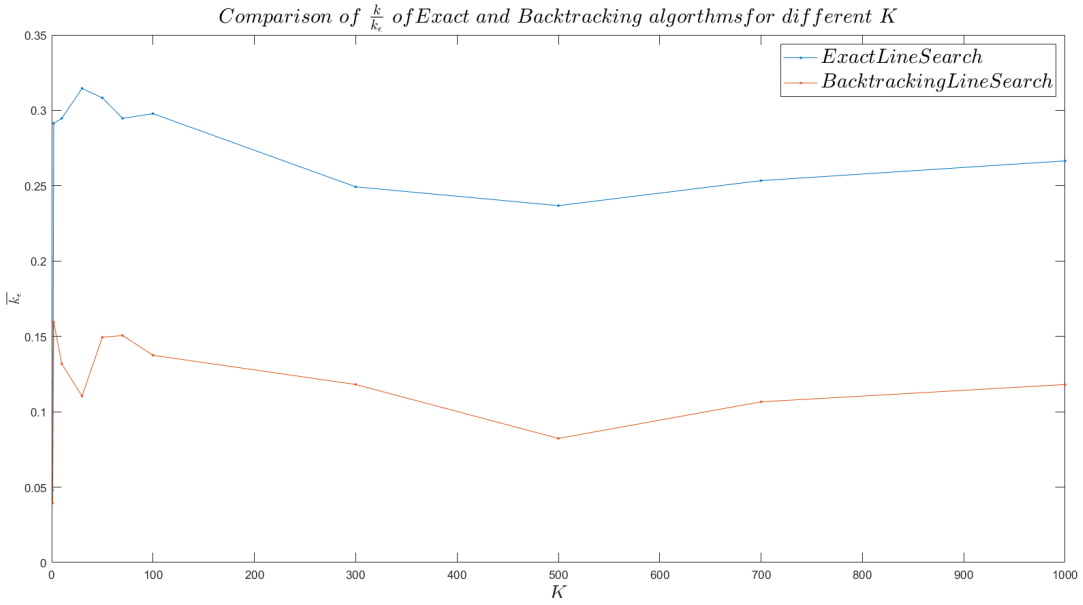


Figure 10: $n = 50$

;

vii. We have already shown that the convergence analysis for both algorithms shows that the minimum number of iterations(k_ϵ) that guarantees solution within accuracy ϵ : $k_\epsilon = \frac{log\left(\frac{f(\mathbf{x_0}-\mathbf{p}^*)}{\epsilon}\right)}{log(\frac{1}{c})}$, where $log(\frac{1}{c}) = \mathbb{K}$ for the exact line search algorithm and $c := 1 - min\{2m\alpha, \frac{2\beta\alpha m}{M}\}$ for the backtracking one. We observe that the ratio $\frac{k}{k_\epsilon}$ for large $\mathbb{R}$ stabilizes and that the algorithms need a lot less iterations to converge than the minimum k_ϵ. Also the exact line tracing algorithm converges faster than the backtracking one.

C Let $\mathbf{c} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^m$ and $\mathbf{A} \in \mathbb{R}^{mxn}$ with rows $\mathbf{a}_i^T, i = 1, ..., m$.
Consider the function $f : \mathbb{R}^n \to \mathbb{R}$, defined as

$$f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} - \sum_{i=1}^{m} log(b_i - \mathbf{a}_i^T\mathbf{x}) = \mathbf{c}^T\mathbf{x} - sum(log(\mathbf{b} - \mathbf{A}\mathbf{x})) \tag{9}$$

with $x \in \mathbb{R}^n$ and $m > n$(or $m >> n$).

- The set $\mathbf{dom}f$ contains only the points $\mathbf{x} \in \mathbb{R}^{\mathbf{n}}$ for which the arguments of the logarithms are positive.

  (a) Let $x_1, x_2 \in domf \Rightarrow \mathbf{b} - \mathbf{A}\mathbf{x_1} > 0$ and $\mathbf{b} - \mathbf{A}\mathbf{x_2} > 0$
  Let $0 < \theta < 1$:

$$\left.\begin{array}{l} \mathbf{b} - \mathbf{A}\mathbf{x_1} > 0 \iff \theta\mathbf{b} - \theta\mathbf{A}\mathbf{x_1} > 0 \\ \mathbf{b} - \mathbf{A}\mathbf{x_1} > 0 \iff (1-\theta)\mathbf{b} - (1-\theta)\mathbf{A}\mathbf{x_1} > 0 \end{array}\right\} \iff$$

$$\theta\mathbf{b} - \theta\mathbf{A}\mathbf{x_1} + (1-\theta)\mathbf{b} - (1-\theta)\mathbf{A}\mathbf{x_1} > 0 \iff \mathbf{b} - \mathbf{A}(\theta\mathbf{x_1} + (\mathbf{1}-\theta)\mathbf{x_2}) < 0 \tag{10}$$

  Therefore $\mathbf{dom}f$ is convex.

  (b) Let $x_1, x_2 \in domf$ and $0 < \theta < 1$:
  The $\nabla f(\mathbf{x})$:

$$\begin{aligned} \nabla f(\mathbf{x}) &= \nabla\mathbf{c}^T\mathbf{x} - \sum_{j=1}^{m} \nabla log(b_j - \mathbf{a}_j^T\mathbf{x}) \\ &= c - \sum_{i=1}^{m} \frac{\nabla(\mathbf{a}_i^T\mathbf{x})}{(b_i - \mathbf{a}_i^T\mathbf{x}} \\ &= c - \sum_{i=1}^{m} \frac{\mathbf{a}_i}{b_i - \mathbf{a}_i^T\mathbf{x}} \end{aligned} \tag{11}$$

  The $\nabla^2 f(\mathbf{x})$ using the chain rule for calculating the hessian:

$$\begin{aligned} \nabla^2 f(\mathbf{x}) &= \nabla^2(\mathbf{c}^T\mathbf{x}) - \sum_{i=1}^{m} \nabla^2 log(b_i - \mathbf{a}_i^T\mathbf{x}) \\ &= -\sum_{i=1}^{m} \frac{1}{b_i - \mathbf{a}_i^T\mathbf{x}} \nabla^2(b_i - \mathbf{a}_i^T\mathbf{x}) - \frac{1}{(b_i - \mathbf{a}_i^T\mathbf{x})^2} \nabla(b_i - \mathbf{a}_i^T\mathbf{x}) \nabla^T(b_i - \mathbf{a}_i^T\mathbf{x}) \\ &= \sum_{i=1}^{m} \frac{\mathbf{a}_i\mathbf{a}_i^T}{(b_i - \mathbf{a}_i^T\mathbf{x})^2} \end{aligned} \tag{12}$$

If the $\nabla^2 f(\mathbf{x})$ is positive definite then f is convex.

Let $z \in \mathbb{R}^m$:

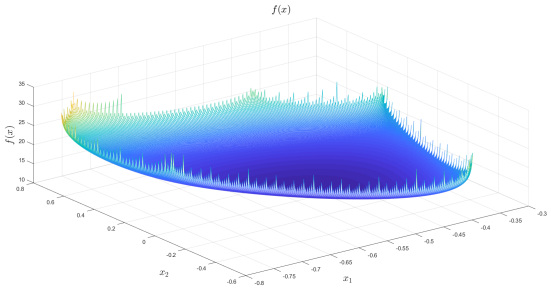
$$\begin{aligned}\mathbf{z}^T \nabla^2 f(\mathbf{x}) \mathbf{z} = \mathbf{z}^T \Big(\sum_{i=1}^{m} \frac{\mathbf{a}_i \mathbf{a}_i^T}{(b_i - \mathbf{a}_i^T \mathbf{x})^2} \Big) \mathbf{z} &= \sum_{i=1}^{m} \frac{\mathbf{z}^T \mathbf{a}_i \mathbf{a}_i^T \mathbf{z}}{(b_i - \mathbf{a}_i^T \mathbf{x})^2} \\ &= \sum_{i=1}^{m} \frac{(\mathbf{a}_i^T \mathbf{z})^T \mathbf{a}_i^T \mathbf{z}}{(b_i - \mathbf{a}_i^T \mathbf{x})^2} \\ &= \sum_{i=1}^{m} \frac{||\mathbf{a}_i^T \mathbf{z}||_2^2}{(b_i - \mathbf{a}_i^T \mathbf{x})^2} > 0 \forall \mathbf{z} \in \mathbb{R}^m - \{\mathbf{0}\}\end{aligned} \tag{13}$$

Therefore $\nabla^2 f(\mathbf{x})$ is positive definite and f is convex.

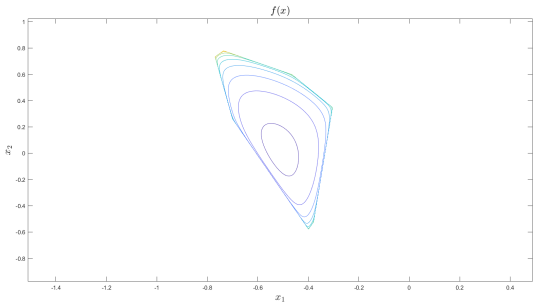
a We minimize f using cvx:

Code:

```
%C.a
cvx_begin
    variable x(n(i),1)
    minimize( c'*x - sum(log(b-A*x)) );
cvx_end
fprintf('Search Time: %e secs\n\n',cvx_cputime);
```



(a) Plot of f

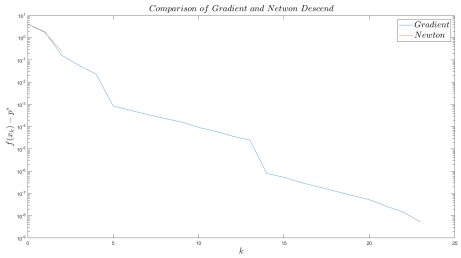


(b) Level Sets of f

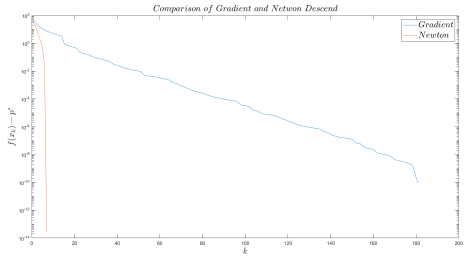
b Since for every point that is not in the domain of f we give it an arbitrarily large value we see spike at the edges of the plot.
Also since the domain of f is the intersection of a finite number of inequalities (in the case only one), domain of f is a polyhedron as seen by the level sets.

c We assume that $\mathbf{x} = \mathbf{0}$ is a feasible point therefore $\mathbf{b} > \mathbf{0}$.
We apply the gradient algorithm using backtracking line search while concurrently checking if we remain in the domain of f.

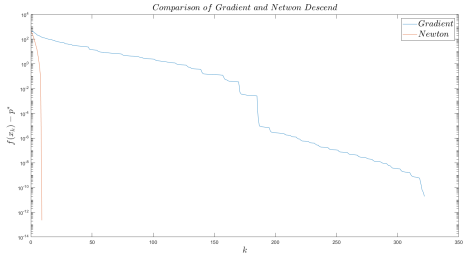
d In order to apply the newton algorithm we calculate the hessian matrix. The exit criterion is $\frac{\lambda^2}{2} \leq \epsilon$ where $\lambda^2 = \nabla^T f(\mathbf{x})(\nabla^{\mathbf{2}}\mathbf{f}(\mathbf{x}))^{-\mathbf{1}}\nabla\mathbf{f}(\mathbf{x})$ and we set the $dx_k = -(\nabla^2 f(\mathbf{x}))^{-\mathbf{1}}\nabla\mathbf{f}(\mathbf{x})$. After we use the backtracking search algorithm while concurrently checking if we remain in the domain of f to find the minima.



(a) $(n, m) = (2, 20)$



(b) $(n, m) = (50, 200)$



(c) $(n, m) = (300, 800)$

e We can see that the newton descend is always faster than the gradient descend. Although the newton algorithm requires significantly more memory because of the need to calculate the hessian.

It is worth noting ,the reason that an algorithm might seem to end prematurely, by looking the semilogy plots, is that since at their last iteration the algorithms find the optimal value within the error ϵ and thus the value $f(\mathbf{x}^*) - p^* = 0$ and the semilogy function cannot draw that. In reality the algorithms have indeed found the optimal value.

Code The matlab code used for the realization of the above experiments. **Exercise2.m**: This script contains the code for section B except subsection B.vii

```
clear; clc; close all;
%% B.i
n=2;

%B.i.a
A = randn(n);
[U,~,~] = svd(A);

disp(U*U'); %Almost equal to I
disp(U'*U);

%B.i.b
l_max = [1,2,10, 100, 1000];
l_min = 1;
fprintf('************************  n=%d  ************************\n
for i=1:size(l_max,2)
    z = l_min + (l_max(i) - l_min) * rand(n - 2, 1);
    eig_v = [l_min ; l_max(i) ; z];
    L = diag(eig_v);

    %B.ii
    CN = l_max(i)/l_min;
    fprintf('
    fprintf('Condition Number = %.0f\n',CN);

    P = U*L*U';
%     chol(P); %Check that P is positive definite
    q = randn(n,1);
    f = @(x) 1/2*x'*P*x + q'*x;
    f2 = @(x1,x2) 1/2*(x1.^2.*P(1,1)+x1.*x2.*P(2,1)+x1.*x2.*P(1,2)+x2
```

```
%B.iii
grad_f = @(x) P*x + q;
%Closed-Form method
x_star_cf = -P\q;
p_star_cf = f(x_star_cf);
%     grad_f(x_star_cf) %Check that grad_f(x_star_cf) is indeed equal
fprintf('Closed-Form Solution: p* = %f',p_star_cf);
fprintf('\n\n');

%B.iv Gradient Descent
epsilon = 10^-5; %Stopping Criterion
x_0 = randn(n,1);
%Exact Line Search
Expected_Iter = CN*log(f((x_0)-p_star_cf)/epsilon);
k_els = 0;
tic;
x_k = x_0;
x_vals_els = x_0;
while norm(grad_f(x_k)) >= epsilon
    t_k = norm(grad_f(x_k))^2/(grad_f(x_k)'*P*grad_f(x_k));
    x_k = x_k - t_k*grad_f(x_k);
    x_vals_els = [x_vals_els x_k];
    k_els = k_els + 1;
end
tEnd=toc;
fprintf('Exact Line Search (Maximum Expected Iterations:%d):\n',c
fprintf('p* = %f\n',f(x_k));
fprintf('x* error MSE = %e\n',mse(x_star_cf, x_k));
fprintf('Iterations: %d\n',k_els);
fprintf('Search Time: %.e secs\n',tEnd);
fprintf('\n');

%Backtracking Line Search
```

```
a = [0.1, 0.2, 0.3, 0.4];
b = [0.1, 0.4, 0.7, 0.9];
k_bls_min = 10^5;
k_bls_max = 0;
figure;
for alpha=1:size(a,2)
    for beta=1:size(b,2)
        Expected_Iter_BLS = log(f((x_0)-p_star_cf)/epsilon)/log(1
        k_bls = 0;
        x_k = x_0;
        tic;
        x_vals_bls = x_0;
        while norm(grad_f(x_k)) >= epsilon
            t_k = 1;
            while f(x_k-t_k*grad_f(x_k)) > f(x_k) - a(alpha)*t_k*
                t_k = b(beta)*t_k;
            end
            x_k = x_k - t_k*grad_f(x_k);
            x_vals_bls = [x_vals_bls x_k];
            k_bls = k_bls + 1;
        end
        tEnd = toc;
        fprintf('Backtracking Search(a=%.2f,b=%.2f, Maximum Expec
        fprintf('p* = %f\n',f(x_k));
        fprintf('x* error MSE = %e\n',mse(x_star_cf, x_k));
        fprintf('Iterations: %d\n',k_bls);
        fprintf('Search Time: %.e secs\n',tEnd);
        fprintf('----------------------------------------------------------------------

        fprintf('\n');
        f_vals_bls = f2(x_vals_bls(1,:),x_vals_bls(2,:));
        if(k_bls > k_bls_max)
            a_max = a(alpha);
```

```
            b_max = b(beta);
            x_vals_bls_max = x_vals_bls;
            f_vals_bls_max = f_vals_bls;
            k_bls_max = k_bls;
        elseif(k_bls < k_bls_min )
            a_min = a(alpha);
            b_min = b(beta);
            x_vals_bls_min = x_vals_bls;
            f_vals_bls_min = f_vals_bls;
            k_bls_min = k_bls;
        end

        %B.vi
        plot(0:k_bls,log(f_vals_bls-p_star_cf));
        hold on;
    end
end
hold off
ylabel('$f(x)-p^{*}$','Interpreter','latex','fontSize',18);
xlabel('$k$','Interpreter','latex','fontSize',18);
title('$Comparison\ of\ Backtracking\ Search\ Value\ Error\ to\ I
legend({'$a=0.1,\ b=0.1$','$a=0.1,\ b=0.4$','$a=0.1, b=0.7$','$a=

%B.v
f = @(x1,x2) 1/2*(x1.^2.*P(1,1)+x1.*x2.*P(2,1)+x1.*x2.*P(1,2)+x2.

% Backtracking Line Search Plots
min_bls_x = min(x_vals_bls(1,:));
max_bls_x = max(x_vals_bls(1,:));
min_bls_y = min(x_vals_bls(2,:));
max_bls_y = max(x_vals_bls(2,:));
[x_1, x_2] = meshgrid(min_bls_x-0.1 : (max_bls_x-min_bls_x)/1000
figure;
```

```
contour(x_1,x_2,f2(x_1,x_2),f_vals_bls_max);
hold on;
plot(x_vals_bls_max(1,:),x_vals_bls_max(2,:),'r.-');
contour(x_1,x_2,f2(x_1,x_2),f_vals_bls_min);
plot(x_vals_bls_min(1,:),x_vals_bls_min(2,:),'b.-');
xlim([min_bls_x-0.1, max_bls_x+0.1]);
ylim([min_bls_y-0.1, max_bls_y+0.1])
hold off;
title('$Comparison\ of\ Fastes\ and\ Slowest\ Parameters\ of\ Bac
fast = sprintf('$a=%.1f, b=%.1f$',a_min,b_min);
slow = sprintf('$a=%.1f, b=%.1f$',a_max,b_max);
legend({'',fast,'',slow},'Interpreter','latex','fontSize',18);
xlabel('$x_1$','Interpreter','latex','fontSize',18);
ylabel('$x_2$','Interpreter','latex','fontSize',18);

% Exact Line Search Plots
f_vals_els = f2(x_vals_els(1,:),x_vals_els(2,:));

min_els_x = min(x_vals_els(1,:));
max_els_x = max(x_vals_els(1,:));
min_els_y = min(x_vals_els(2,:));
max_els_y = max(x_vals_els(2,:));
[x_1, x_2] = meshgrid(min_els_x-0.1 : (max_els_x-min_els_x)/1000
figure;
contour(x_1,x_2,f2(x_1,x_2),f_vals_els);
hold on;
plot(x_vals_els(1,:),x_vals_els(2,:),'.-');
xlim([min_els_x-0.1, max_els_x+0.1]);
ylim([min_els_y-0.1, max_els_y+0.1])
hold off;
title('$Exact\ Line\ Search$','Interpreter','latex','fontSize',18
xlabel('$x_1$','Interpreter','latex','fontSize',18);
ylabel('$x_2$','Interpreter','latex','fontSize',18);
```

```
    %B.vi
    figure;
    plot(0:k_els,log(f_vals_els-p_star_cf),'.-');
    title('$Exact\ Line\ Search\ Value\ Error\ to\ Iteration$','Inter
    ylabel('$f(x)-p^{*}$','Interpreter','latex','fontSize',18);
    xlabel('$k$','Interpreter','latex','fontSize',18);
end
```

Exercise2_Bvii.m: This script contains the code for section B.vii.

```
clear; clc; close all;
%% B.i
n=50;

%B.i.a
A = randn(n);
[U,~,~] = svd(A);

%B.i.b
l_max = [1,2,10,30,50,70,100,300,500,700,1000];
l_min = 1;
k_frac_els=zeros(1,size(l_max,2));
k_frac_bls=zeros(1,size(l_max,2));
for i=1:size(l_max,2)
    z = l_min + (l_max(i) - l_min) * rand(n - 2, 1);
    eig_v = [l_min ; l_max(i) ; z];
    L = diag(eig_v);

    %B.ii
    CN = l_max(i)/l_min;

    P = U*L*U';
%     chol(P); %Check that P is positive definite
```

```
q = randn(n,1);
f = @(x) 1/2*x'*P*x + q'*x;

%B.iii
grad_f = @(x) P*x + q;
%Closed-Form method
x_star_cf = -P\q;
p_star_cf = f(x_star_cf);

%B.iv Gradient Descent
epsilon = 10^-5; %Stopping Criterion
x_0 = randn(n,1);
%Exact Line Search
Expected_Iter = CN*log(f((x_0)-p_star_cf)/epsilon);
k_els = 0;
tic;
x_k = x_0;
while norm(grad_f(x_k)) >= epsilon
    t_k = norm(grad_f(x_k))^2/(grad_f(x_k)'*P*grad_f(x_k));
    x_k = x_k - t_k*grad_f(x_k);
    k_els = k_els + 1;
end
k_frac_els(i)=k_els/Expected_Iter;
%Backtracking Line Search
a = 0.4;
b = 0.7;
Expected_Iter_BLS = log(f((x_0)-p_star_cf)/epsilon)/log(1/(1-( mi
k_bls = 0;
x_k = x_0;
while norm(grad_f(x_k)) >= epsilon
    t_k = 1;
    while f(x_k-t_k*grad_f(x_k)) > f(x_k) - a*t_k*grad_f(x_k)'*gr
        t_k = b*t_k;
```

```
        end
        x_k = x_k - t_k*grad_f(x_k);
        k_bls = k_bls + 1;
    end
    k_frac_bls(i)=k_bls/Expected_Iter_BLS;
end
plot(l_max,k_frac_els,'.-');
hold on;
plot(l_max,k_frac_bls,'.-');
title('$Comparison\ of\ \frac{k}{k_\epsilon}\ of Exact\ and\ Backtrac
legend({'$Exact Line Search$','$Backtracking Line Search$'},'Interpre
xlabel('$K$','Interpreter','latex','fontSize',18);
ylabel('$\frac{k}{k_\epsilon}$','Interpreter','latex','fontSize',18);
```

Exercise2_C.m: This script contains the code for section C.

```
clear; clc; close all;
%% C

n = [2, 50, 300];
m = [20, 200, 800];

for i = 1 : 3
    fprintf('----------------[n=%d , m=%d]----------------',n(i),m(i));
    A = randn(m(i), n(i));
    b = rand(m(i),1);
    c = randn(n(i),1);

    %C.a
    cvx_begin
        variable x(n(i),1)
        minimize( c'*x - sum(log(b-A*x)) );
    cvx_end
    fprintf('Search Time: %e secs\n\n',cvx_cputime);
```

```
%C.b
log_arg =@(x) b - A*x;
if n(i) == 2
    x1 = cvx_optpnt.x(1)-1 : 2/1000 : cvx_optpnt.x(1)+1;
    x2 = cvx_optpnt.x(2)-1 : 2/1000 : cvx_optpnt.x(2)+1;
    f = zeros(size(x1,2),size(x1,2));

    for k = 1:size(x1,2)
        for j = 1:size(x2,2)
            lg_arg = log_arg( [x1(k) ; x2(j)] );
            if min(lg_arg) >= 0
                f(k,j) = c'*[x1(k);x2(j)] - sum(log(lg_arg));
            else
                f(k,j) = -inf;
            end
        end
    end
    figure;
    mesh(x1,x2,f);
    title('$f(x)$','Interpreter','latex','fontSize',18);
    xlabel('$x_1$','Interpreter','latex','fontSize',18);
    ylabel('$x_2$','Interpreter','latex','fontSize',18);
    zlabel('$f(x)$','Interpreter','latex','fontSize',18);

    figure;
    contour(x1,x2,f);
    title('$f(x)$','Interpreter','latex','fontSize',18);
    xlabel('$x_1$','Interpreter','latex','fontSize',18);
    ylabel('$x_2$','Interpreter','latex','fontSize',18);
    zlabel('$f(x)$','Interpreter','latex','fontSize',18);
end
```

```
%C.c
f = @(x) c'*x - sum(log(b-A*x));
epsilon = 10^-3; %Stopping Criterion
x_0 = zeros(n(i),1);
alpha = 0.5;
beta = 0.7;
k_grad = 0;
x_k = x_0;

grad = grad_f(x_k,m(i),A,b,c);
tic;
x_vals_grad = x_0;
while norm(grad) >= epsilon
    t_k = 1;
    while min(log_arg(x_k-t_k*grad)) < 0
        t_k = beta*t_k;
    end

    while f(x_k-t_k*grad) > f(x_k) - alpha*t_k*grad'*grad
        t_k = beta*t_k;
    end
    x_k = x_k - t_k*grad;

    grad = grad_f(x_k,m(i),A,b,c);
    x_vals_grad = [x_vals_grad x_k];
    k_grad = k_grad + 1;
end
tEnd=toc;
fprintf('Gradient Descend with Bactracking Line Search\n');
x_opt_grad = x_k;
optval_grad = f(x_k);
fprintf('p* = %f\n',optval_grad);
fprintf('x* error MSE with cvx= %e\n',mse(cvx_optpnt.x, x_k));
```

```
fprintf('p* error MSE with cvx= %e\n',mse(cvx_optval, optval_grad
fprintf('Iterations: %d\n',k_grad);
fprintf('Search Time: %.e secs\n',tEnd);
fprintf('\n');

%C.d
k_newt = 0;
x_k = x_0;
x_vals_newt = x_0;
tic;
while(1)
    hessian = hessian_f(x_k,A,b,n(i),m(i));
    grad = grad_f(x_k,m(i),A,b,c);

    lambda2 = grad'/hessian*grad;
    if(lambda2/2 <= epsilon)
        break;
    end

    dx = -hessian\grad;
    t_k = 1;
    while min(log_arg(x_k+t_k*dx)) < 0
        t_k = beta*t_k;
    end

    while f(x_k+t_k*dx) > f(x_k) + alpha*t_k*grad'*dx
        t_k = beta*t_k;
    end
    x_k = x_k + t_k*dx;
    x_vals_newt = [x_vals_newt x_k];
    k_newt = k_newt+1;
end
tEnd=toc;
```

```
    fprintf('Newton Descend with Bactracking Line Search\n');
    x_opt_newt = x_k;
    optval_newt = f(x_k);
    fprintf('p* = %f\n',optval_newt);
    fprintf('x* error MSE with cvx= %e\n',mse(cvx_optpnt.x, x_k));
    fprintf('p* error MSE with cvx= %e\n',mse(cvx_optval, optval_newt
    fprintf('Iterations: %d\n',k_newt);
    fprintf('Search Time: %.e secs\n',tEnd);
    fprintf('\n');

    %C.e
    figure;
    semilogy(0:k_grad,f(x_vals_grad)-optval_grad);
    hold on;
    semilogy(0:k_newt,f(x_vals_newt)-optval_newt);
    hold off;
    title('$Comparison\ of\ Gradient\ and\ Netwon\ Descend$','Interpr
    xlabel('$k$','Interpreter','latex','fontSize',18);
    ylabel('$f(x_k)-p^*$','Interpreter','latex','fontSize',18);
    legend({'$Gradient$','$Newton$'},'Interpreter','latex','fontSize'
end

function h = hessian_f(x,A,b,n,m)
    h = zeros(n,n);
    for i = 1:n
        for j = 1:m
            h(:,i) = h(:,i)+A(j,i)*A(j,:)'/ (b(j)-A(j,:)*x)^2;
        end
    end

end
function g = grad_f(x,m,A,b,c)
    g = c;
```

```
    for  l = 1:m
        g = g + A(l,:)'/(b(l)-(A(l,:)*x));
    end
end
```