Convex Optimization

Exercise 4

Report Delivery Date: 29 May 2018

Instructor: Athanasios P. Liavas

Student: Konidaris Vissarion 2011030123

In this exercise we solve the following optimization problem with inequality constraints in three different ways. First by cvx, then using the barrier method starting from a feasible point and at last with the Primal-Dual algorithm. Let the convex optimization problem

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & f_0(x) = c^T x \\ \text{subject to} \quad & Ax - b = 0 \\ & x \succeq 0, \end{aligned}$$

with $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{p\times n}$ and $b \in \mathbb{R}^p$. We create the parameters c, A and b in a random way, while guaranteeing that a feasible solution to the problem actually exists.

1. At first we solve the problem using the cvx library(in Matlab code below).

2. We proceed by solving the problem using the barrier method starting from a feasible point.

   (a) First we need to find a feasible point, or else, a point that belongs to the set $\mathbb{X} := \{x \in \cap_{i=0}^{n} \boldsymbol{dom} f_i \quad | \quad f_i(x) \le 0, i = 1, ..., m, \quad \boldsymbol{Ax} = \boldsymbol{b}\}$. We find such a point by solving the convex problem.

   $$\begin{aligned} \underset{x\in\mathbb{R}^n, s\in\mathbb{R}}{\text{minimize}} \quad & s \\ \text{subject to} \quad & Ax - b = 0 \\ & -x \preceq s \end{aligned}$$

   In order to solve the above feasibility problem we use the barrier method and solve the sequence of problems

   $$\begin{aligned} \underset{x\in\mathbb{R}^n, s\in\mathbb{R}}{\text{minimize}} \quad & ts - \sum_{i=1}^{n} log(x_i + s) \\ \text{subject to} \quad & Ax - b = 0. \end{aligned}$$

We stop the algorithm as soon as $s < 0$. There is no need to find the optimal solution.

(b) Starting from the feasible point that we got from the feasibility problem, we then solve our original problem using the barrier method. That is, we solve the sequence of problems

$$\begin{aligned} \underset{x \in \mathbb{R}^n, s \in \mathbb{R}}{\text{minimize}} \quad & tc - \sum_{i=1}^{n} log(x_i) \\ \text{subject to} \quad & Ax - b = 0. \end{aligned}$$

The barrier method algorithm and the Newton step of the problem can be seen below.

$\boldsymbol{x} \in \boldsymbol{dom} f_0 \cap \boldsymbol{dom} \phi$, $\boldsymbol{Ax} = \boldsymbol{b}$, $t > 0$, $\mu > 1$, tolerance $\epsilon > 0$

while $TRUE$ **do**

1. Compute $x_*(t)$, by minimizing $tf_0 + \phi$ subject to $\boldsymbol{Ax} = \boldsymbol{b}$, starting at x.
2. $x := x_*(t)$.
3. quit if $\frac{m}{t} < \epsilon$.
4. $t := \mu t.$.

end

Algorithm 1: Barrier method for convex optimization problems (starting from a feasible point).

$$w = -\left(A\left(\nabla^2 f(x)\right)^{-1} A^T\right)^{-1} A\left(\nabla^2 f(x)\right)^{-1} \nabla f(x)$$

$$\Delta_{x_{N_t}} = -\left(\nabla^2 f(x)\right)^{-1}\left(\nabla f(x) + A^T w\right)$$

$$\lambda^2(x) = -\nabla f(x)^T \Delta_{x_{N_t}}$$

$$\nabla f(x) = tc - \begin{bmatrix} -x_1^{-1} & \ldots.. & -x_n^{-1} \end{bmatrix}^T$$

$$\nabla^2 f(x) = diag(\begin{bmatrix} x_1^{-2} & \ldots.. & x_n^{-2} \end{bmatrix}^T).$$

3. Finally we solve the same problem sequence using the Primal-Dual algorithm. The algorithm and the Newton step can be seen below.

$\boldsymbol{x} \in \textbf{\textit{dom}} f_0 \cap \textbf{\textit{dom}} \phi$, $\lambda > 0$, $\mu > 1$, ϵ_{feas}, $\epsilon > 0$

repeat

1. $t := \frac{m\mu}{\hat{n}}$.
2. Compute $\Delta \boldsymbol{y_{pd}}$
3. Perform line search and choose step s.
4. $\boldsymbol{y} := \boldsymbol{y} + s\Delta \boldsymbol{y_{pd}}$

until $(||r_t||_2 < \epsilon_{feas}, \quad \hat{n} < \epsilon)$;

Algorithm 2: Primal-Dual algorithm for convex optimization problems.

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(x) \\ : \\ f_n(x) \end{bmatrix}, D\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \nabla f_1(x)^T \\ : \\ \nabla f_n(x)^T \end{bmatrix}$$

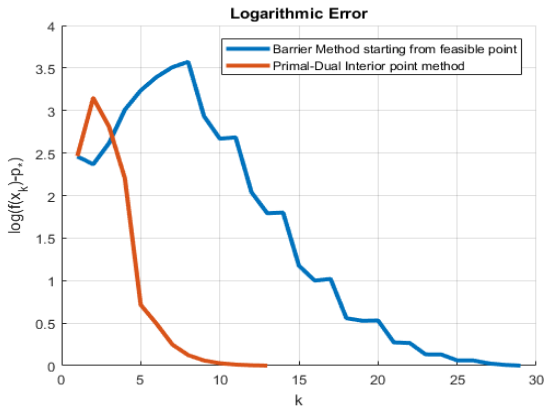
$$r_t(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{\lambda}) = \begin{bmatrix} r_{t,d}(\boldsymbol{y}) \\ r_{t,c}(\boldsymbol{y}) \\ r_{t,p}(\boldsymbol{y}) \end{bmatrix} = \begin{bmatrix} \nabla f_0(\boldsymbol{x}) + D\boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\lambda} + \boldsymbol{A}^T \boldsymbol{v} \\ -\textbf{\textit{diag}}(\boldsymbol{\lambda})\boldsymbol{f}(\boldsymbol{x}) - \frac{1}{t}\mathbf{1} \\ \boldsymbol{Ax} - \boldsymbol{b} \end{bmatrix}$$

$$\begin{bmatrix} \nabla^2 f_0(\boldsymbol{x}) + \sum_{i=1}^n \lambda_i \nabla^2 f_i(\boldsymbol{x}) & D\boldsymbol{f}(\boldsymbol{x})^T & \boldsymbol{A}^T \\ -\textbf{\textit{diag}}(\boldsymbol{\lambda} D\boldsymbol{f}(\boldsymbol{x})) & -\textbf{\textit{diag}}(\boldsymbol{f}(\boldsymbol{x})) & \boldsymbol{O} \\ \boldsymbol{A} & \boldsymbol{O} & \boldsymbol{O} \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{\lambda} \\ \Delta \boldsymbol{v} \end{bmatrix} = - \begin{bmatrix} r_{t,d}(\boldsymbol{y}) \\ r_{t,c}(\boldsymbol{y}) \\ r_{t,p}(\boldsymbol{y}) \end{bmatrix}$$

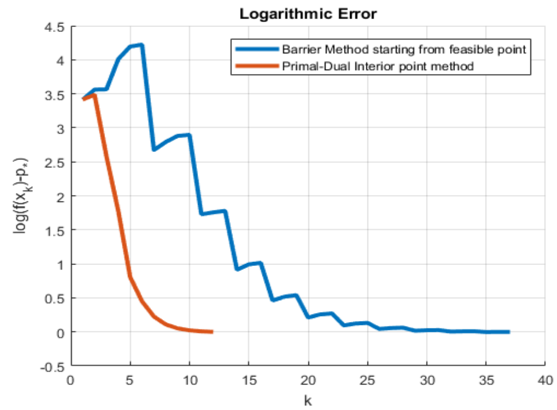
$$\hat{n}(\boldsymbol{x}, \boldsymbol{\lambda}) := -\boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\lambda}$$

The quantity $\nabla^2 f_0(\boldsymbol{x}) + \sum_{i=1}^n \lambda_i \nabla^2 f_i(\boldsymbol{x})$ is equal to a zero n by n matrix. Lastly, $\boldsymbol{f}(\boldsymbol{x}) = -\boldsymbol{x}$ and $D\boldsymbol{f}(\boldsymbol{x}) = D\boldsymbol{f}(\boldsymbol{x})^T = -I_{n\times n}$.

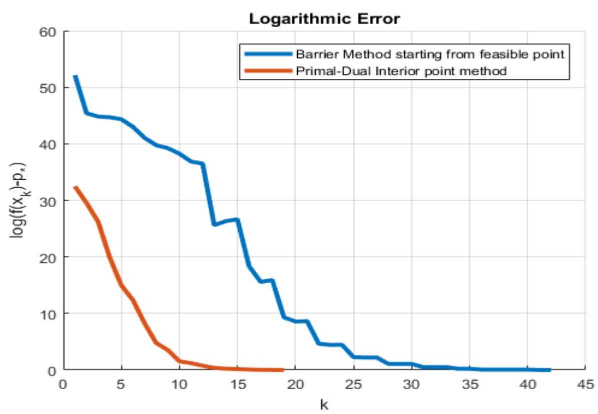
Taking a look at the plots below we can get a pretty good idea of the convergence rate of the two algorithms. The Primal-Dual algorithm clearly converges faster. The trials were made with various α and β parameters for the backtracking algorithm. The ones that worked the best were the pairs (0.01, 0.5) and (0.3, 0.8).



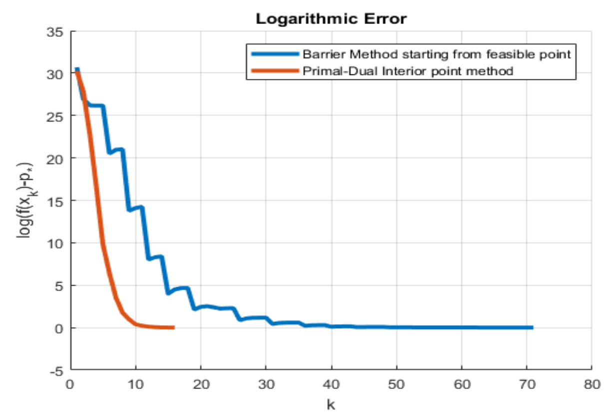
(a) $n = 50$, $p = 2$, $\epsilon = 0.001$



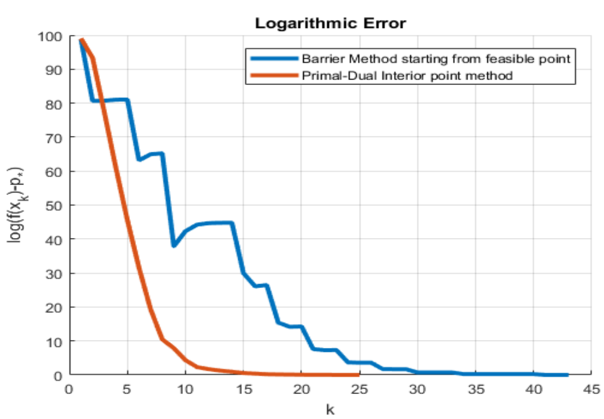
(b) $n = 50$, $p = 2$, $\epsilon = 0.0001$



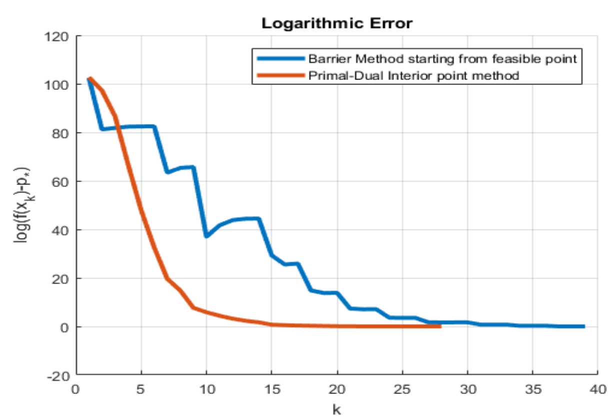
(c) $n = 200$, $p = 50$, $\epsilon = 0.001$



(d) $n = 200$, $p = 50$, $\epsilon = 0.0001$



(e) $n = 800$, $p = 300$, $\epsilon = 0.001$



(f) $n = 800$, $p = 300$, $\epsilon = 0.0001$

Figure 1: Plots of the quantity $log(f_0(x_k) - f_0^*)$ for the two algorithms.

Matlab implementation

Main.m

```
breaklines
% Vissarion Konidaris
% Convex Optimization Course
% 22/5/2018

clear ; close all; clc
format long;

disp('Minimization problem');
disp('dot(c,x)');
disp('subject to x>=0 and Ax=b');

in=0;
while(in~=1 && in~=2 && in~=3)
  disp('Give the dimensions of the problem.');
  disp('Indicative value pairs are (n,p) = (50,2), ');
  disp('(200,500),(800,300)\nwhere m the number of the ');
  disp('logarithms and n the dimension of input x.');
  disp('Type 1 for (50,2), 2 for (200,50), 3 for (800,300).');
  in=input('n : ');
  if(in==1)
    n=20;
    p=2;
  elseif(in==2)
    n=200;
    p=50;
  else
    n=800;
    p=300;
  end
  disp('');
end

% The acceptable error of the optimizer.
e = -1.;
while(e<0.)
```

```
  disp('Give the acceptable error of the optimizer.');
  disp('Acceptable number of error are ');
  disp('the all the positive real numbers.');
  e=input('e : ');
  disp('');
end

alpha=0.5;
beta=1.;
while(alpha>=0.5 || alpha<=0.)
  disp('Give a value for the parameter aplha.');
  disp('Acceptable values are in the open interval (0,0.5).');
  alpha=input('aplha : ');
  disp('');
end
while(beta>=1. || beta<=0.)
  disp('Give a value for the parameter beta.');
  disp('Acceptable values are in the open interval (0,1).');
  beta=input('beta : ');
  disp('');
end

A=rand(p,n);
x=rand(n,1);
c=rand(n,1);
b=A*x;


%%% Cvx solution %%%
%%%%%%%%%%%%%%%%%%%%

cvx_begin
    variable cv_x(n);
    minimize (c'*cv_x);
    subject to
        A*cv_x-b==0;
        -cv_x<=0;
cvx_end
cv_min_fun_val=c'*cv_x;
```

```
%%%%%%%%% Deasibility problem %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pseudoinverseA=pinv(A);
x0=pseudoinverseA*b;

s=max(-x0)+1;
mu=2;
tau=2;
feasible=0;
iter=0;
A_=[A zeros(size(A,1),1)];
while(1)
    while(1)
        temp = -(x0+s).^-1;
        grad_phi = [temp; sum(temp)];
        clear temp;
        grad = [zeros(n,1);  tau] + grad_phi;
        temp = (x0+s).^-2;
        hessian_phi = diag( [temp; sum(temp)] );
        hessian_phi(n+1,1:n) = temp';
        hessian_phi(1:n,n+1) = temp;
        clear temp;
        hessian_inv=pinv(hessian_phi);
        clear clear temp;

        v=-inv(A_*hessian_inv*A_')*A_*hessian_inv*grad;
        delta=-hessian_inv*(grad+A_'*v);
        lambda_squared=-grad'*delta;

        if(lambda_squared/2<=e)
            break;
        end

        t=1;
        while(1)
            in_domain=1;
            point = [x0; s] + t*delta;
            for i=1:n
```

```
                if(point(i,1)+point(n+1,1)<=0)
                    in_domain=0;
                    clear point;
                    break;
                end
            end
            if(in_domain==0)
                disp(['Not in domain at loop ',num2str(iter+1)]);
                t=beta*t;
                clear point;
                continue;
            end
            break;
        end
        f=tau*s-sum(log(x0+s));
        while( tau*(s+t*delta(n+1,1))-sum( log( (x0+t*delta(1:n,1))+...
                (s+t*delta(n+1,1)) ) )> f-alpha*t*lambda_squared)
            t=beta*t;
        end

        iter=iter+1;
        x0=x0+t*delta(1:n,1);
        s=s+t*delta(n+1,1);

        % Stopping earlier.
        if(s<0)
            feasible=1;
            break;
        end
    end

    if((mu/tau)<e || feasible==1)
        break;
    end
    tau=tau*mu;
end



%%%%%% Barrier method starting from feasible point %%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mu = 2;
tau = 2;
iter = 0;
x1 = x0;
traj1 = [x1];
while(1)
    while(1)
        grad = tau*c-x1.^-1;
        hessian_inv=inv(diag(x1.^-2));
        v=-inv(A*hessian_inv*A')*A*hessian_inv*grad;
        delta=-hessian_inv*(grad+A'*v);
        lambda_squared=-grad'*delta;

        %disp(norm(cv_x-x))

        if(lambda_squared/2<=e)
            break;
        end

        t=1;
        while(1)
            in_domain=1;
            point = x1+t*delta;
            for i=1:size(point,1)
                if(point(i,1)<=0)
                    feasible=0;
                    clear point;
                    break;
                end
            end
            if(in_domain==0)
                disp(['Not in domain at loop ',num2str(iter+1)]);
                t=beta*t;
                clear point;
                continue;
            end
            break;
        end
```

```
        f=tau*c'*x1-sum(log(x1));
        while( tau*c'*(x1+t*delta)-sum(log(x1+t*delta) )...
            > f-alpha*t*lambda_squared)
            t=beta*t;
        end

        iter=iter+1;
        x1=x1+t*delta;
        traj1 = [traj1 x1];
    end

    if((mu/tau)<e)
        break;
    end
    tau=tau*mu;
end

%%%%%% Barrier method starting from infeasible point %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mu=20;
tau=2;
epsilon_feas=0.001;
x2=x0;
l=rand(n,1);
v=rand(p,1);
en = x2'*l;
rho = [c-1*diag(ones(n,1))*l+A'*v;...
       diag(l)*x2-ones(n,1)*tau^-1;...
       A*x2-b];
norm_rho =norm(rho);
tau=mu*p/en;
traj2=[x2];
while(norm_rho>=epsilon_feas && en>=e)

    mat=zeros(2*n+p,2*n+p);
    mat(1:n,n+1:2*n) = -1*diag(ones(n,1));
    mat(1:n, 2*n+1:2*n+p) = A';
    mat(n+1:2*n,1:n) = diag(l)*diag(ones(n,1));
    mat(n+1:2*n,n+1:2*n) = diag(x2);
```

```
mat(2*n+1:2*n+p,1:n) = A;

deltas = -inv(mat)*rho;

% Determinig step s.
s=1;
for i=n+1:2*n
    if(deltas(i,1)<0 && -l(i-n,1)/deltas(i,1)<s)
        s=-l(i-n,1)/deltas(i,1);
    end
end

% Backtracking s until f(x+)<0.
s=0.99*s;
while(1)
    feas=1;
    point = x2+s*deltas(1:n,1);
    for i=1:n
        if( -point(i,1) >=0 )
            feas=0;
            break;
        end
    end
    if(feas==1)
        break;
    end
    s=beta*s;
end

rtd_plus=c-1*diag(ones(n,1))*(l+s*deltas(n+1:2*n,1))+...
          A'*(v+s*deltas(2*n+1:2*n+p,1));
rtc_plus=diag((l+s*deltas(n+1:2*n,1)))*(x2+s*deltas(1:n,1))-...
          ones(n,1)*tau^-1;
rtp_plus=A*(x2+s*deltas(1:n,1))-b;
rho_plus=[rtd_plus;rtc_plus;rtp_plus];
norm_rho_plus=norm(rho_plus);
while(norm_rho_plus^2 >(1-alpha*s)*norm_rho^2)
  s=beta*s;
  rtd_plus=c-1*diag(ones(n,1))*(l+s*deltas(n+1:2*n,1))+...
            A'*(v+s*deltas(2*n+1:2*n+p,1));
```

```matlab
            rtc_plus=diag(l+s*deltas(n+1:2*n,1))*(x2+s*deltas(1:n,1))-...
                     ones(n,1)*tau^-1;
            rtp_plus=A*(x2+s*deltas(1:n,1))-b;
            rho_plus=[rtd_plus;rtc_plus;rtp_plus];
            norm_rho_plus=norm(rho_plus);
        end
        x2 = x2+s*deltas(1:n,1);
        l = l+s*deltas(n+1:2*n,1);
        v = v+s*deltas(2*n+1:2*n+p,1);

        en = x2'*l;
        tau=mu*p/en;
        rho = [c-1*diag(ones(n,1))*l+A'*v;...
               diag(l)*x2-ones(n,1)*tau^-1;...
               A*x2-b];
        norm_rho = norm(rho);
        traj2 = [traj2 x2];

end

v1=zeros(1,size(traj1,2));
for i=1:size(traj1,2)
    v1(1,i)=c'*traj1(:,i);
end
iterations1 = linspace(1,size(traj1,2),size(traj1,2));

v2=zeros(1,size(traj2,2));
for i=1:size(traj2,2)
    v2(1,i)=c'*traj2(:,i);
end
iterations2 = linspace(1,size(traj2,2),size(traj2,2));

figure(1);hold on;
semilogy(iterations1,v1-v1(1,size(v1,2)),'linewidth',3);
semilogy(iterations2,v2-v2(1,size(v2,2)),'linewidth',3);
title('Logarithmic Error');
xlabel('k');
ylabel('log(f(x_k)-p_*)');
legend('Barrier Meth from feasible point','Primal-Dual Interior Point meth');
grid on;
```

```
disp(['Optimal value (cvx_optval): ',...
      num2str(c'*cv_x)]);
disp(['Optimal value (Barrier Mathod starting from feasible point): ',...
      num2str(c'*x1)]);
disp(['Optimal value (Primal-Dual Interior Point Method): ',...
      num2str(c'*x2)]);
```