Thibault Laugel – MVA

Convex Optimization Exercises 3 – 24/11/2014

Exercise 1

We focus on the following problem :

$$\min_{x \in \boldsymbol{R}^n} f(x) := \sum_{i=1}^{m} \log(b_i - a_i^T x)$$

With $A \in \boldsymbol{R}^{mxn}$ and $b \in \boldsymbol{R}^m$.

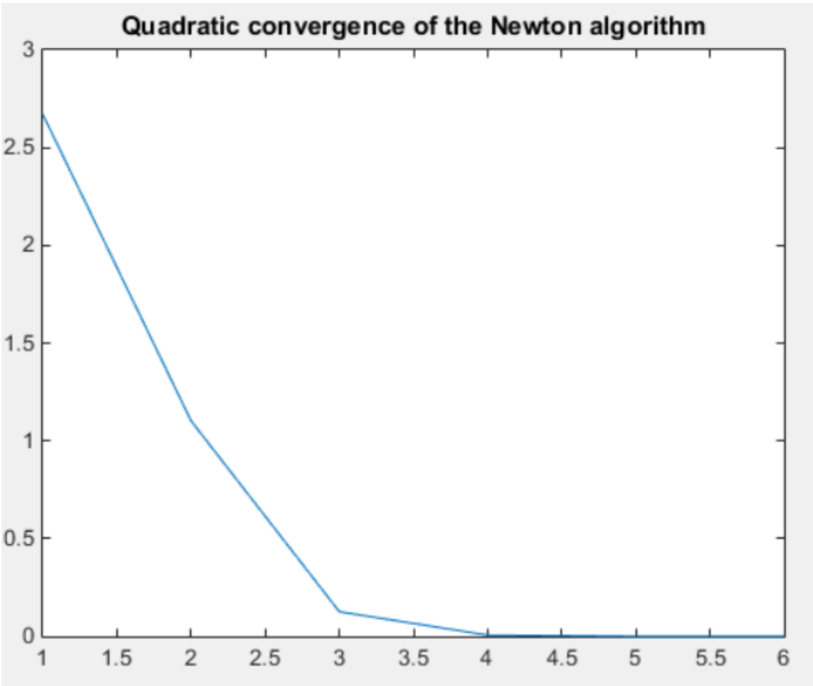
The following code implements Newton's method with backtracking line search to compute the minimum of the desired function :

```
%Newton's method with backtracking line search for the desired function

function [x,i,lambda, f] = Newton1(A, b, maxIter, threshold, alpha, beta)
%initialization
[m,n] = size(A);
x = zeros(n, maxIter);

%begin iteration loop
for i = 1:maxIter
    %gradient and hessian
    f = -sum(log(b-A*x(:,i)));
    d = 1./(b - A * x(:,i));
    grad = A' * d ;
    Hess = A' * diag(d.^2) * A;
    %Newton step and decrement
    Dx = - inv(Hess)*grad;
    lambda = - grad' * Dx;

    %stopping creterion
    if (lambda / 2) < threshold
        break
    else
        t = 1;
        %backtracking line search
        while -sum(log(b-A*(x(:,i)+t*Dx))) > (f+alpha*t*grad'*Dx)
            t = beta * t;
        end

        %update
        x(:,i+1) = x(:,i) + t * Dx;
    end
end
end
```

We test the code with the following parameters values :

```
%% Initialisation :
% Function parameters
A = [-rand(1)*5, 0.5; rand(1)*7, -2*rand(1); rand(1), rand(1); -1, -1];
b = [1; 2; 3 ; 4];
% Newton's method parameters :
maxIter = 1000;
alpha = 0.1;
beta = 0.5;
threshold = 1e-10;
```

We then plot the convergence and get the following graph :



We can indeed see a quadratic convergence.

Exercise 2

We focus on the following problem :

$$\min_{s.t.\ Ax \leq b} f(x) \coloneqq c^T x$$

The following code implements the logarithmic barrier method with backtraking line search to compute the minimum of the desired function :

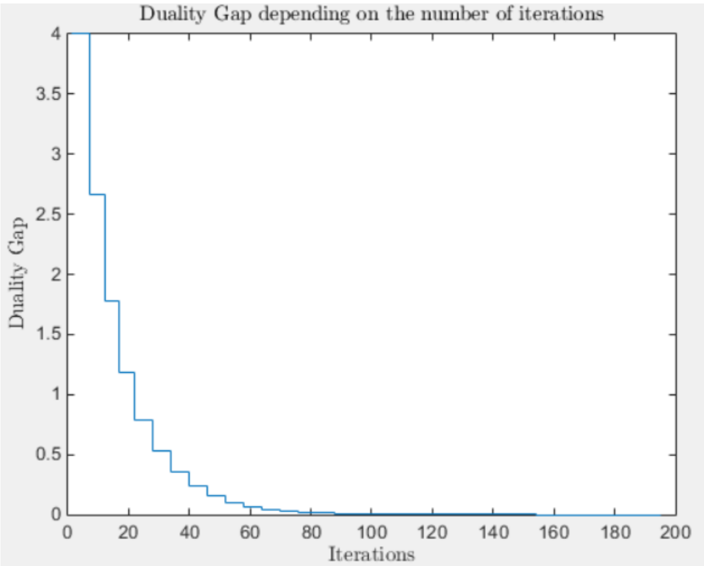
```
%%initialization
%interior points parameters
thresholdIP = 1e-5 %threshold for interior point method
mu = 1.5
x0 = [0, 0]'
T = 1
dualityGap = zeros(1, maxIter) ; %track duality gap
T_values = zeros(1, maxIter) ; %track values of T parameter
N_Iterations = [] ; %used for the plotting
prev_i = 0 ; %used for the plotting
ord = [] ; %used for the plotting

%%solve the program using Interior Point method with Newton + backtracking
at each step
for j = 1:maxIter
    T_values(1,j) = T ;
    [x,i,lambda2, objVal, dualObjVal, dualArg] =newton2(A, b, c, T, x0,
maxIter, tol, alpha, beta) ;
    dualityGap(:,j) = objVal - dualObjVal ;
    N_Iterations = [N_Iterations, (1:i) + prev_i] ; %for plotting
    prev_i = prev_i + i ; %for plotting
    ord = [ord, repmat(dualityGap(:,j), 1, i)] ; %for plotting
    if (dualityGap(:,j) < thresholdIP)
        break;
    else
        T = mu*T;
    x0 = x(:,i);
    end;
end;


%%newton method
function [x,i,lambda, f, dualf, dualArg] = newton2(A, b, c, T, x0, maxIter,
threshold, alpha, beta)
%compute initial parameters
[m,n] = size(A);
x = zeros(n, maxIter);
x(:,1) = x0;

%begin iteration loop
for i = 1:maxIter
    %gradient and hessian
    f = T*c'*x(:,i) - sum(log(b-A*x(:,i)));
    d = 1./(b - A * x(:,i));
    grad = T*c + A' * d;
    Hess = A' * diag(d.^2) * A;
```

```
    %Newton step and decrement
    Dx = - Hess \ grad;
    lambda = - grad' * Dx;

    %stopping criterion
    if (lambda / 2) < threshold
        dualf = f -m/T;
        dualArg = 1/T * d;
        break
    else
        %backtracking line search
        t = 1;
        while T*c'*(x(:,i)+t*Dx) -sum(log(b-A*(x(:,i)+ t*Dx))) >
(f+alpha*t*grad'*Dx)
            t = beta * t ;
        end
        %update
        x(:,i+1) = x(:,i) + t * Dx ;
    end
end
end
```

We test the code with the following parameters values (same as before) :

```
%Initialisation:
A = [-rand(1)*5, 0.5; rand(1)*7, -2*rand(1); rand(1), rand(1); -1, -1];
b = [1; 2; 3 ; 4];
c = [1 ; 1]
T = 1
%Newton's method parameters
maxIter = 150;
tol = 1e-10;
alpha = 0.1;
beta = 0.5;
```

The following graph represenst the duality gap obtained with our algorithm :



Finally, in order to have a function that computes a strictly feasible solution to $Ax \leq b$ if there is one, or returns an error message if not, we simply write the following code to replace our previous Newton function in our algorithm:

```
function [x,i,lambda, f, dualf, dualArg] = newton3(A, b, c, T, x0, maxIter,
threshold, alpha, beta)
    [m,n] = size(A);
    if sum(A*x0 < b) == m %feasibility condition
        [x,i,lambda, f, dualf, dualArg] = newton2(A, b, c, T, x0, maxIter,
threshold, alpha, beta);
    else
        error('Error : x0 is not a strictly feasible point') %return ERROR
message
    end
end
```