Christopher Tsai
March 4, 2008
EE 364 – Convex Optimization

CVX Problem Set VIII – Standard Form LP Barrier Method

Problem #1 – Centering Step

We first implement Newton's Method for solving the analytic centering problem

$$minimize \;\; c^T x - \sum_{i=1}^{n} \ln x_i$$

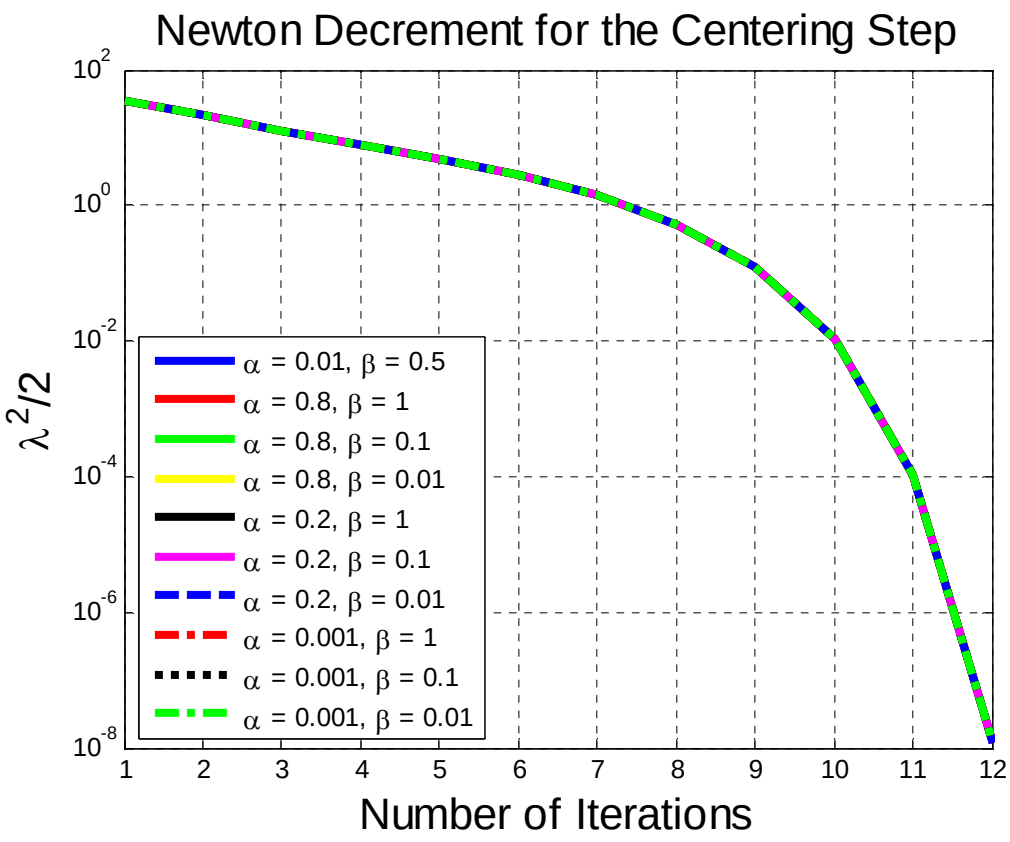
$$subject\ to \;\; Ax = b$$

The following function accepts A, b, c, and starting point x_0 to compute the primal optimal point $x^\star$, the dual optimal point $\nu^\star$, and the Newton decrement computed as $\frac{\lambda^2}{2}$.

```
function [x, v, nSteps] = NewtonCentering (A, b, c, x0, alpha, beta)
%
% [X, V, NSTEPS] = NEWTONCENTERING (A, B, C, X0, ALPHA, BETA)
%
% ------------------------------------------------------------------------
%
% Taps Newton's Method to solve the analytic centering problem:
%
%                       minimize    c'x - sum(ln(x))
%                       subject to  Ax = b
%
% ------------------------------------------------------------------------
%
% INPUTS:
% -------
% A -- (m x n) coefficient matrix in constraint
% b -- (m x 1) vector in constraint
% c -- (n x 1) coefficient vector in objective
% x0 -- (n x 1) vector defining initial starting point in Newton's Method
% alpha -- scalar constant, affects convergence
% beta -- scalar constant, affects convergence
%
% OUTPUTS:
% --------
% x -- primal optimal point
% v -- dual optimal point
% nSteps -- number of Newton steps executed, contains lambda^2/2
%
% ------------------------------------------------------------------------

[m,n] = size(A);

% Setting default backtracking parameters:
if (nargin < 5)
    alpha = 0.01;
    beta = 0.5;
elseif (nargin < 6)
    beta = 0.5;
end

% Termination criteria:
tolerance = 1e-6;
maxIterations = 100;
i = 0;
```

```
if ( (norm(A*x0 - b) > 1e-3) || (min(x0) <= 0) )
    fprintf('\nYou have chosen an ominous starting point!  Aborting now. . .\n');
    x = [];
    v = [];
    nSteps = [];
    return;
else

    % Beginning Newton's Method:
    x = x0;
    nSteps = [];

    for i = 1 : maxIterations
        hessian = diag(1./x.^2);
        gradient = c - 1./x;

        % Employing block elimination to compute the Newton step:
        y = [hessian, A.' ; A, zeros(m,m)] \ [-gradient ; zeros(m,1)];
        dx = y(1:n);
        v = y(n+1:end);

        % Stopping criterion based on lambda^2:
        lambda2 = -gradient' * dx;
        nSteps = [nSteps lambda2/2];
        if (lambda2/2 <= tolerance)
            break;
        end

        % BACKTRACKING LINE SEARCH:
        t = 1;

        while (min(x + t*dx) <= 0)
            t = beta*t;
        end

        % Incremental step modification:
        while (c.'*(t*dx) - sum(log(x + t*dx)) + sum(log(x)) > alpha*t*gradient'*dx)
            t = beta*t;
        end

        % Advancing time:
        x = x + t*dx;

    end
end

% ----------------------------------------------------------------------
```

Generating random feasible data, we simulate the centering step and observe the decrement gradually converge to zero over more and more iterations, until convergence is complete around 10 iterations. Note from the graph that α and β have little to no noticeable effect on either the optimal value or the convergence process; the Newton steps are indistinguishable on a logarithmic scale, all decaying to zero at the same rate:



As we expect, the Newton decrement decreases quadratically. Our algorithm behaves properly.

Finally, we verify that KKT conditions hold when our solver yields a valid primal optimal point:

```
% Checking that our computed x* and v* satisfy the KKT conditions:
primalFeasibility = (abs(A*x-b) <= 1e-4);
primalPositivity = (x >= 0);
dualFeasibility = (A.'*v + c >= 0);
complementarity = (abs(A*x-b).*v <= 1e-4);
gradLagrangian = (abs(2*x + c + A.'*v) <= 1e-4);
```

Indeed, our KKT conditions hold for optimal values.

Problem #2 – LP Solver with Strictly Feasible Starting Point

We are now prepared to solve the standard form linear program (LP) with *two* constraints: an inequality constraint Ax = b in addition to the positivity constraint on the primal optimal point:

$$minimize \;\; c^T x$$

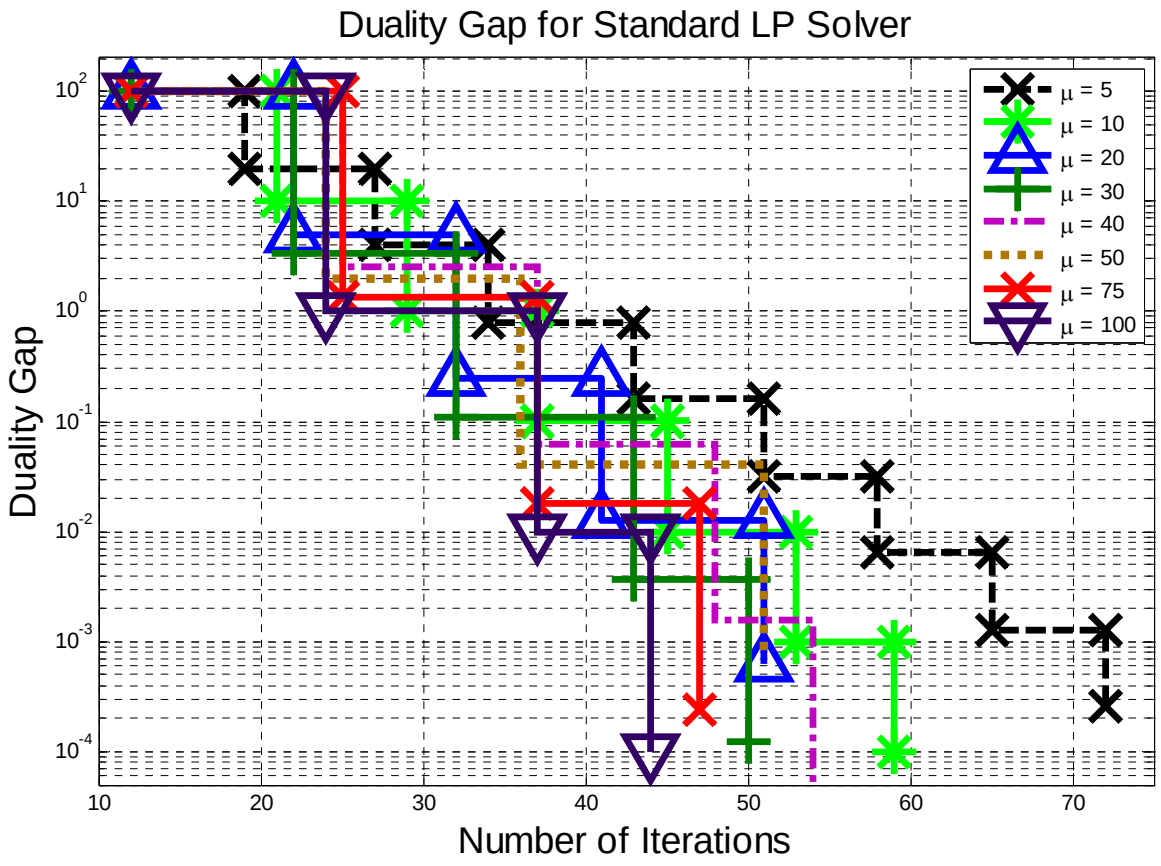
$$subject\ to \;\; Ax = b, \qquad x \succcurlyeq 0$$

We can use the centering code to compute the optimal point iteratively until the duality gap $\frac{n}{t}$ has sufficiently decreased to smaller than 10^{-3}. Thus, our wrapper function assumes the form:

```
function [x, v, history] = StandardLPBarrier (A, b, c, x0, mu)
%
% [X, V, HISTORY] = STANDARDLPBARRIER (A, B, C, X0, MU)
%
% ------------------------------------------------------------------------
%
% Taps Newton's Method to solve the standard form LP:
%
%                       minimize    c'x
%                       subject to  Ax == b
%                                    x >= 0
%
% ------------------------------------------------------------------------
%
% INPUTS:
% -------
% A -- (m x n) coefficient matrix in constraint
% b -- (m x 1) vector in constraint
% c -- (n x 1) coefficient vector in objective
% x0 -- (n x 1) vector defining initial starting point in Newton's Method
% mu -- scalar constant, affects barrier method
%
% OUTPUTS:
% --------
% x -- primal optimal point
% v -- dual optimal point
% history -- data concerning the convergence of the algorithm
%         history(1,:) contains the number of iterations per centering
%         history(2,:) contains the (ideally) closing duality gap
%
% ------------------------------------------------------------------------

[m,n] = size(A);

% Setting barrier method parameters:
if (nargin < 5)
    mu = 50;
end

tolerance = 1e-3;
t0 = 1;

% Initialization, preparing to begin:
x = x0;
t = t0;
history = [];

% Iterating and solving until the duality gap is tolerably low:
while (true)
    [xOpt, vOpt, nIterations] = NewtonCentering (A, b, t*c, x);
```

```
    x = xOpt;

    % Computing the current duality gap:
    dualityGap = m/t;

    % Updating the history:
    history = [history [length(nIterations) ; dualityGap]];

    % Checking termination criterion . . .
    % STOP when duality gap is sufficiently small:
    if (dualityGap < tolerance)
        break;
    end

    % Otherwise, advance . . .
    t = mu*t;
    v = -vOpt/t;
end

% ------------------------------------------------------------------------
```

Our LP solver checks affirmatively with `CVX` for several problem instances, returning an infeasible tag for infeasible problems and an optimal point for feasible problems. Meanwhile, we can tweak the parameter μ to vary the rate at which the duality gap closes; unlike α and β, the value of μ closely governs the convergence rate of the duality gap, which we can witness in the number of iterations:



As the plot reveals, increasing the value of μ decreases the number of iterations while lengthening the number of iterations at the same duality gap. The ultimate duality gap is a matter of chance; depending on where a time step falls, another step may not be necessary. We see that, coincidentally, the largest value of μ also coincides with the smallest duality gap and the fastest convergence (only 44 iterations).

We solve a sample problem with both a professional solver (CVX) as well as our homemade LP solver, juxtaposing responses to ascertain their proper functionality:

```
% Solving a sample linear program to ensure that cvx agrees:
cvx_begin;

    variable X(n);
    minimize (c.'*X)
    subject to
        A*X == b;
        X >= 0;

cvx_end;

% CVX yields an optimal value of ...37.032532!
% Our Standard LP Barrier Method yields an optimum of ...37.032632!
```

Indeed, our solver also fails when CVX fails, as for infeasible problems:

```
% INFEASIBLE problem:
A = randn(m - 1, n);
A = [A ; ones(1,n)];
b = rand(m,1);

cvx_begin;

    variable X(n);
    minimize (c.'*X)
    subject to
        A*X == b;
        X >= 0;

cvx_end;
[x, v, history] = StandardLPBarrier (A, b, c, x0, 50);

…

cvx:
Status: Infeasible
Optimal value (cvx_optval): +Inf

StandardLPBarrier: >> You have chosen an ominous starting point!  Aborting now. . .
```

Problem #3 – LP Solver

In order to determine a logical feasible first step to our barrier method algorithm, we must solve the Phase I problem:

$$minimize \ \ t$$

$$subject\ to \ \ Ax = b,$$

$$x \succcurlyeq (1-t)\vec{\mathbf{1}}$$

$$t \geq 0$$

The benefit to first solve the Phase I problem is its ability to shed the positivity constraint on x, allowing us to begin the feasibility problem with *any* feasible x. We choose, for instance, the least-norm solution to our underdetermined ($m < n$) system.

We can reformulate the Phase I problem as a standard form LP, which we can readily and swiftly solve with the same barrier method that we will implement to solve the actual standard form LP problem itself:

$$minimize\ \tilde{c}^T\tilde{z} = \begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix}^T \begin{bmatrix}t\\z_1\\\vdots\\z_n\end{bmatrix}$$

$$subject\ to\ \tilde{A}\tilde{z} = \tilde{b}\ and\ \tilde{z} \succcurlyeq \vec{\mathbf{0}}$$

$$\begin{bmatrix}-\mathrm{A}\vec{\mathbf{1}} & \mathrm{A}\end{bmatrix}\begin{bmatrix}t\\z_1\\\vdots\\z_n\end{bmatrix} = \begin{bmatrix}b_1 - a_1^T\\b_2 - a_2^T\\\vdots\\b_m - a_m^T\end{bmatrix},$$

$$\begin{bmatrix}t\\z_1\\\vdots\\z_n\end{bmatrix} \succcurlyeq \begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix}$$

Thus, by creating a new vector with t as the first (or last) element, and augmenting the coefficient matrix to accommodate t, we can solve a standard form LP on the vector of z. Once we

find a feasible z, and t, we can recover a feasible x to insert into our barrier method by reverting our variable: $x = z - (t-1) = z + 1 - t$. If any x is feasible, then this x will be feasible, providing us an initial point with which to begin our Newton's Method-based barrier method to solve the primal LP as follows:

```
function x = LPSolver (A, b, c)
%
% X = LPSOLVER (A, B, C)
%
% ------------------------------------------------------------------------
%
% Taps Newton's Method to solve the standard form LP:
%
%                       minimize    c'x
%                       subject to  Ax == b
%                                    x >= 0
%
% ------------------------------------------------------------------------
%
% INPUTS:
% -------
% A -- (m x n) coefficient matrix in constraint
% b -- (m x 1) vector in constraint
% c -- (n x 1) coefficient vector in objective
%
% OUTPUTS:
% --------
% x -- primal optimal point
%
% ------------------------------------------------------------------------

[m,n] = size(A);
mu = 20;

% Devising a potential initial point for the Phase I Problem:
x0 = pinv(A)*b
t0 = 2 + max(0,-min(x0))

% Forming the relevant matrices for the Phase I Feasibility LP:
rowSums = -A*ones(n,1);
APhase = [rowSums, A];
bPhase = b-A*ones(n,1);
cPhase = [1 ; zeros(n,1)];
z0 = [t0 ; (x0 + (t0-1)*ones(n,1))];

% Solving the Phase I Feasibility LP for a valid initial point:

[zFeas, vFeas, past] = StandardLPBarrier (APhase, bPhase, cPhase, z0, mu);
if (cPhase'*zFeas >= 1)
    error('Standard Form LP is INFEASBIBLE!');
    return;
end
z = zFeas(2:end);
t = zFeas(1);
x0 = z-(t-1)*ones(n,1);

% If FEASBIBLE (x,t) found....
    % Solving the primal problem:
[x, v, history] = StandardLPBarrier (A, b, c, x0);


% ------------------------------------------------------------------------
```

We devise a few simple test cases to verify that our linear program solver works:

```
% ------------------------------------------------------------------------
m = 1;
n = 10;
c = (1:n).';
A = ones(1,n);
b = 1;

% CVX RESPONSE:
% Status: Solved
% Optimal value (cvx_optval): +1

% LP SOLVER RESPONSE:
% optVal = 1.0001

% ------------------------------------------------------------------------

m = 100;
n = 200;
xOpt = (1:n).';
randn('state',1);
A = randn(m,n);
randn('state',2);
b = A*xOpt;
c = (rand(n,1));

% CVX RESPONSE:
% Status: Solved
% Optimal value (cvx_optval): +5650.73

% LP SOLVER RESPONSE:
% optVal = 5650.7274

% ------------------------------------------------------------------------

m = 100;
n = 500;
randn('state',1);
A = randn(m,n);
randn('state',2);
x0 = 5 + randn(n,1);
b = A*x0;
c = 2*(rand(n,1));

% CVX RESPONSE:
% Status: Solved
% Optimal value (cvx_optval): +6.25366

% LP SOLVER RESPONSE:
% optVal = 6.25367

% ------------------------------------------------------------------------

m = 100;
n = 500;
randn('state',1);
A = randn(m-1,n);
A = [A ; ones(1,n)];
randn('state',2);
b = randn(m,1);

% CVX RESPONSE:
% Status: Infeasible
% Optimal value (cvx_optval): +Inf

% LP SOLVER RESPONSE:
% ??? Error using ==> LPSolver
% Standard Form LP is INFEASBIBLE!
% ------------------------------------------------------------------------
```