EE 159/CS 168 - Convex Optimization

Scott Fullenbaum

Final Exam

---

1. (a) Consider the problem individually. Without loss of generality, we define the function $g(x,y) = xlog(x/y)$ where $x, y \in \mathbb{R}^+$, so we're looking at two individual terms. To show $g(x,y)$ is convex, first we can see that $\nabla g(x,y) = [log(x/y) + 1, -x/y]^T$. Then, we can derive the Hessian and get $\nabla^2 g(x,y) = \begin{bmatrix} 1/x & -1/y \\ -1/y & x/y^2 \end{bmatrix}$. From here, we can use Sylvester's Criterion and on $\mathbb{R}_+$, $1/x > 0$. Additionally, the determinant of $\nabla^2 g(x,y) = 0$. Therefore, $\nabla^2 g(x,y)$ is Positive Semi-Definite by Sylvester's Criterion, and $g(x,y)$ is convex. Since $f(x,y)$ is the sum of n versions of $g(x,y)$ with different variables, and convex functions are preserved under sums, then $f(x,y)$ is convex and therefore jointly convex in x and y.

   (b) i. First, we introduce dual variables $\lambda \in \mathbb{R}^n$ and $\nu \in \mathbb{R}$. Then, we can formulate the Lagrangian:
   $\mathcal{L}(x,\lambda,\nu) = \sum_{k=1}^{n} x_k log(x_k/y_k) + \lambda^T(b - Ax) + \nu(1 - 1^T x)$
   $\frac{\partial}{\partial x_i}\mathcal{L}(x,\lambda,\nu) = 1 + log(x_i/y_i) - a_i^T\lambda - \nu = 0$ and this implies that at $\frac{\partial \mathcal{L}}{\partial x_i} = 0$ at $x_k = y_k e^{a_k^T\lambda + \nu - 1}$. We can then plug in this value back into $\mathcal{L}(x,\lambda,\nu)$ to get the dual function:
   $g(\lambda,\nu) = \sum_{k=1}^{n} y_k e^{a_k^T\lambda+\nu-1} log(y_k e^{a_k^T\lambda+\nu-1}/y_k) + \lambda^T(b - Ax) + \nu(1 - 1^T x)$. Assume x is a vector formed by individual terms derived above. It's a bit too much to type out.
   $g(\lambda,\nu) = \sum_{k=1}^{n} y_k e^{a_k^T\lambda+\nu-1}(a_k^T\lambda + \nu - 1) + \lambda^T(b - Ax) + \nu(1 - 1^T x)$. From here, we distribute out the terms and get:
   $g(\lambda,\nu) = \sum_{k=1}^{n} y_k e^{a_k^T\lambda+\nu-1} a_k^T\lambda + \sum_{k=1}^{n} y_k e^{a_k^T\lambda+\nu-1}\nu - \sum_{k=1}^{n} y_k e^{a_k^T\lambda+\nu-1} + \lambda^T b - \lambda^T Ax + \nu - \nu 1^T x$.
   First, since $1^T x = \sum_{i=1}^{n} x_i$, then $-v1^T x = -\sum_{k=1}^{n} y_k e^{a_k^T\lambda+\nu-1}\nu$. We can reach a sim-

ilar conclusion with $-\lambda^T Ax$ meaning this term cancels out with the first sum. So, $g(\lambda,\nu) = \lambda^T b + \nu - \sum_{k=1}^{n} y_k e^{a_k^T \lambda + \nu - 1}$ and the dual problem is:

$$\begin{aligned} &\text{maximize } \lambda^T b + \nu - \sum_{k=1}^{n} y_k e^{a_k^T \lambda + \nu - 1} \\ &\text{such that } \lambda \succeq 0 \end{aligned}$$

To simplify further, we differentiate the dual function with respect to ν. $\frac{\partial}{\partial \nu} g(\lambda,\nu) = 1 - \sum_{k=1}^{n} y_k e^{a_k^T \lambda + \nu - 1} = 0$. So, have the equation, $\sum_{k=1}^{n} y_k e^{a_k^T \lambda + \nu - 1} = 1$. Since $\nu - 1$ is constant, can rewrite the equation as: $e^{\nu - 1} \sum_{k=1}^{n} y_k e^{a_k^T \lambda} = 1$. Then we can divide both sides and isolate ν and get $\nu = 1 - log \sum_{k=1}^{n} y_k e^{a_k^T \lambda}$

Then, we plug this value for ν back into the equation. So: $g(\lambda) = b^T \lambda + 1 - log \sum_{k=1}^{n} y_k e^{a_k^T \lambda} - \sum_{k=1}^{n} y_k e^{a_k^T \lambda - log(\sum_{k=1}^{n} y_k e^{a_k^T \lambda})}$. We can see this long sum on the right simplifies to 1, so then $g(\lambda) = b^T \lambda - log \sum_{k=1}^{n} y_k e^{a_k^T \lambda}$. The dual problem is then:

$$\begin{aligned} &\text{maximize } b^T \lambda - log \sum_{k=1}^{n} y_k e^{a_k^T \lambda} \\ &\text{such that } \lambda \succeq 0 \end{aligned}$$

ii. Assume the (x, λ, ν) are optimal, then the KKT conditions are:

$[1 + log(x_1/y_1), 1 + log(x_2/y_2), ..., 1 + log(x_n/y_n)]^T + \lambda^T A + \nu = 0$

$\lambda \succeq 0$

$Ax - b \preceq 0$

$1^T x - 1 = 0$

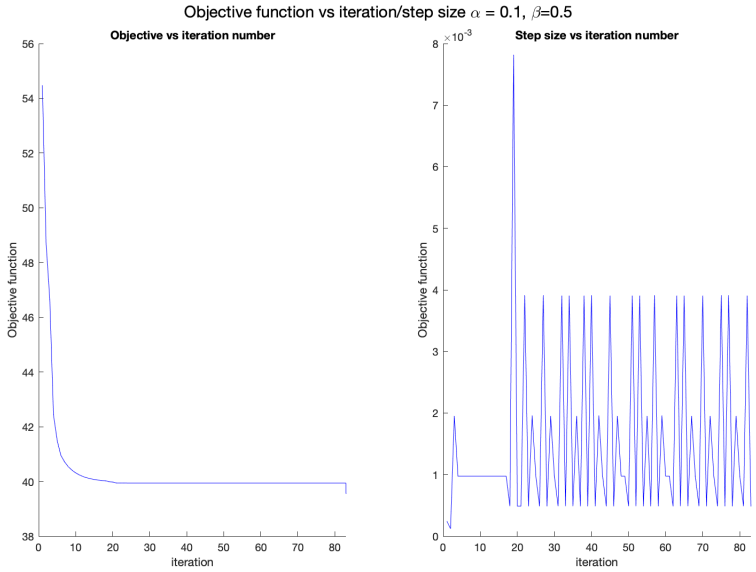
$\lambda^T (Ax - b) = 0$

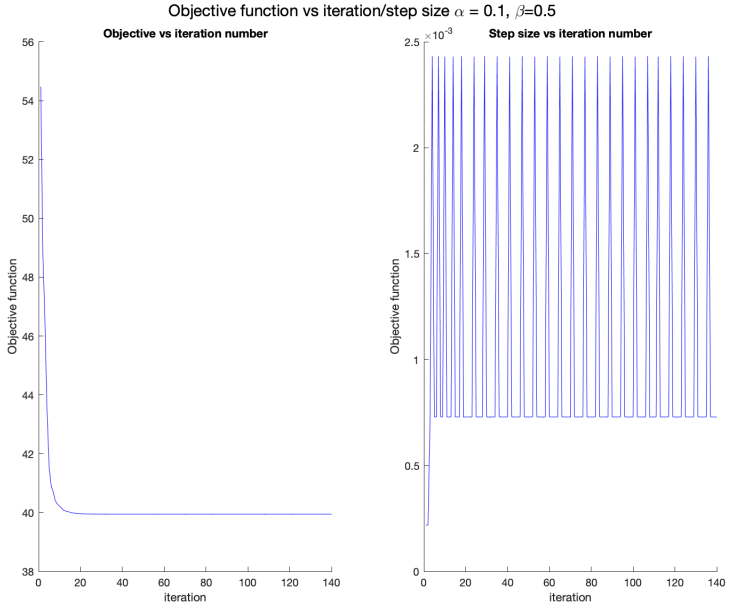
2. (a) The Hessian and Gradient of the log part of the equation were partially derived for the last homework, just sub out the previous answer of 1 for b. If we call the function $f(x)$, then $\nabla f(x) = x + A^T(\frac{1}{b - Ax})$. Additionally, $\nabla^2 f(x) = A^T diag(\frac{1}{b - Ax})^2 A + 1$. For gradient descent, the search direction at $x \in \mathbf{dom} f$ is $-\nabla f(x) = -x - A^T(\frac{1}{b - Ax})$. For Newton's method the search direction $x_{nt} = -nabla^2 f(x)^{-1} \nabla f(x)$. There doesn't

appear to be a nice simplification of this step, so to get just plug in the Hessian and gradient above.

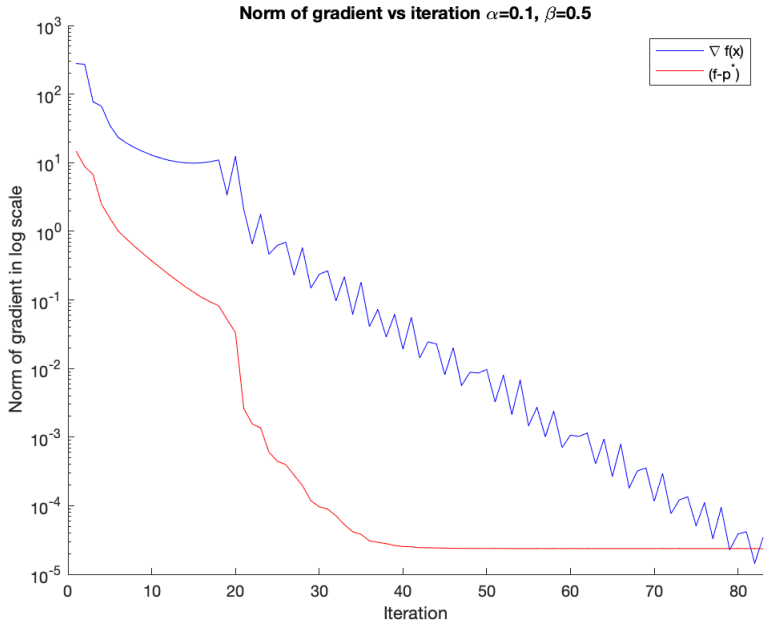
(b) Implementation of Gradient descent is at the end. For the following part, I've just put the plots for both methods together.

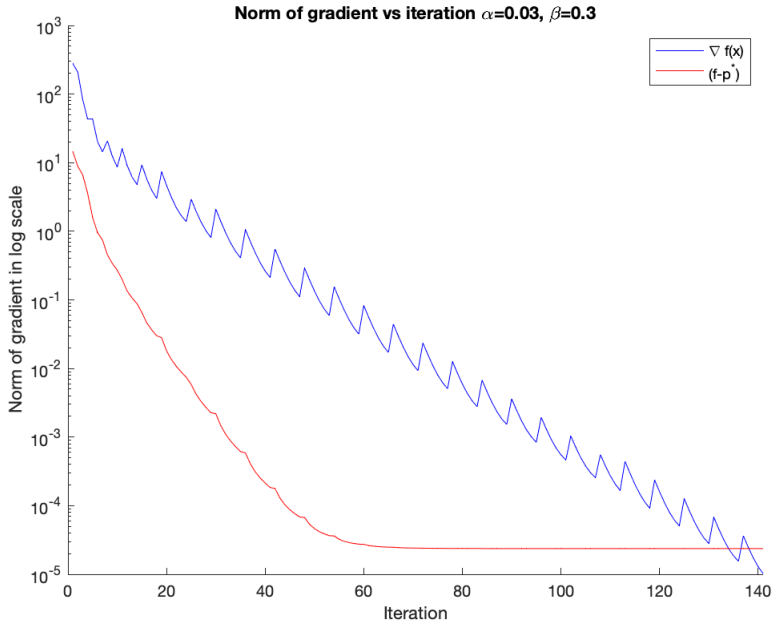
i. For the objective function vs iteration number. If the plot is too big it goes off screen, but code reproduces it:





ii. The two graphs below combine part ii and iii. I found that $p^* = 39.430$ after trying different values of α and β.



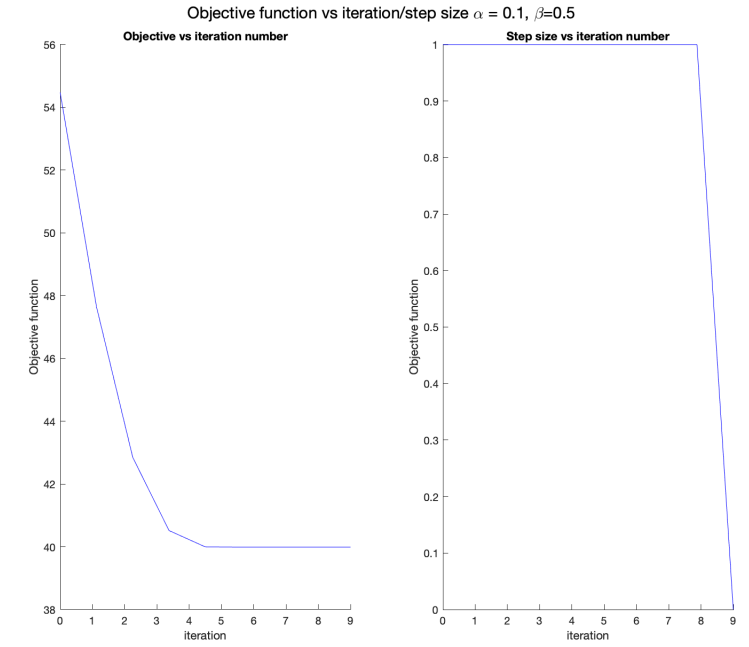


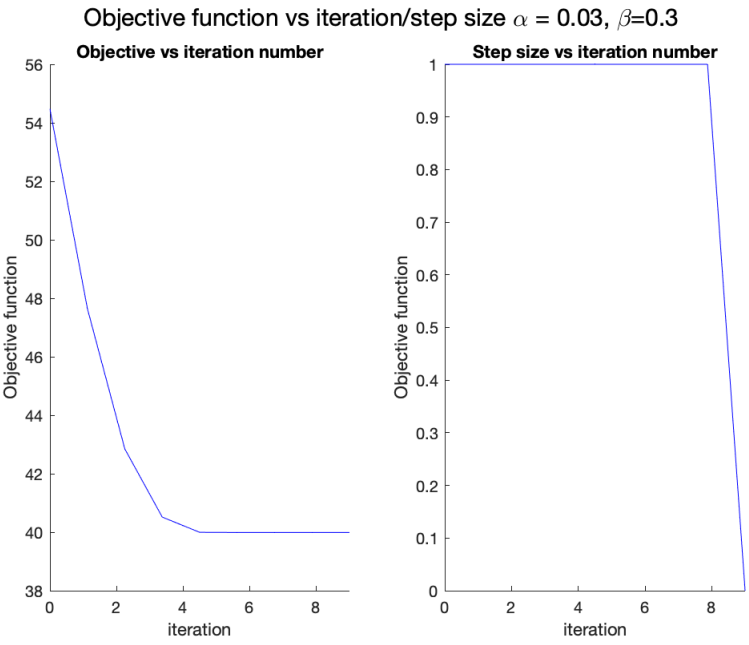
iii. These graphs make sense, as we can see at the end we get below the tolerance bounds and then stop. However, we converge much faster for $\alpha = 0.1, \beta = 0.5$. We can also see that the graph of $(f - p^*)$ for the second case appears to level

off faster, while the first red graph appears to make a couple of bigger jumps. Additionally the gap between $f - p^*$ appears to converge much faster to the optimal solution and then levels off. I believe a heuristic stopping condition will lead to good accuracy. To prove, let f be a convex function, and x and y denote two successive steps in gradient descent, so $f(x) \geq f(y)$. By the first order condition of convexity, $f(x) \geq f(y) + \nabla f(x)^T(x - y)$. We can rearrange this condition into $\nabla f(x)^T(x - y) \leq f(x) - f(y)$. Under a heuristic stopping criteria, we are checking if $f(x) - f(y) < \epsilon$ so the equation becomes $\nabla f(x)^T(x - y) < \epsilon$. By Cauchy-Schwarz, $\nabla f(x)^T(x-y) \leq \|\nabla f(x)\|\|x-y\|$. Therefore, we can conclude $|\nabla f(x)\| < \epsilon/(\|x-y\|)$. So, by satisfying the heuristic stopping criteria, we are satisfying the $\|\nabla f(x)\|$, though at a slightly higher tolerance level, since x and y are probably close together, so $\|x - y\|$ is small, causing ϵ to increase. This lines up with our result above, as the objective stops changing after a much shorter set of iterations. Because of this a heuristic stopping condition seems to be effective, but you would probably pick a lower ϵ as it seems to reach the tolerance bound faster.

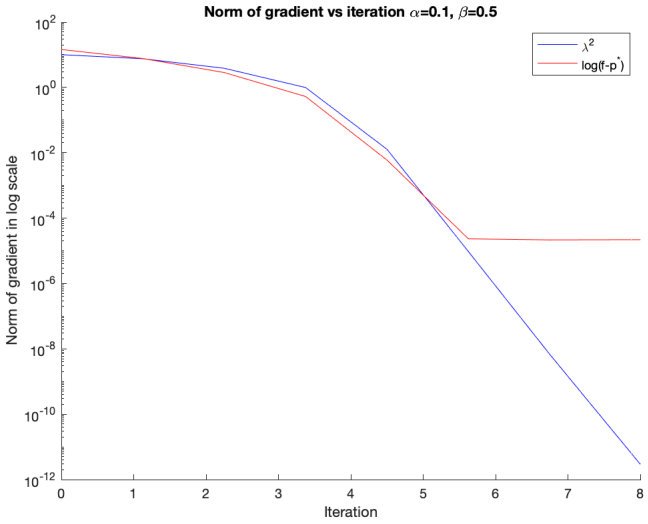
(c) Newton's method. One thing I want to note, is no matter how I tweaked α and β for Newton's method, I couldn't get the optimal value from gradient descent and have $p^* = 39.9898$. I'm guessing this is some kind of numerical stability or accuracy issue with either method, since these matrices can be ill-conditioned.

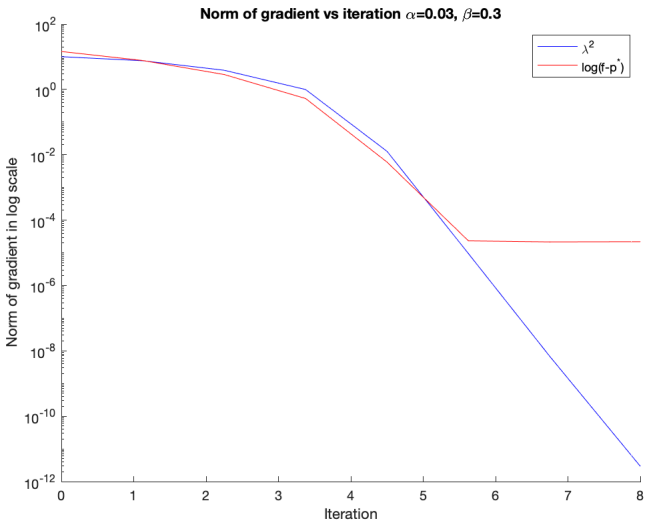
i. Here are both graphs on the next page:





ii. For the second version with log scales, here are the graphs:





They do appear to match up, I did verify that they were both operating under the correct α and β. We can see that they are nearly identical for the most part but then the gap between the two levels off a bit faster. So, for the last couple iterations it's hovering around the same point just changing slightly to meet the decrement.

3. (a) i. The KKT conditions are as follows, assume $\lambda, \nu \in \mathbb{R}^n$:

$$c + \lambda + A^T\nu = 0$$

$$\lambda^T x = 0$$

$$\lambda \succeq 0$$

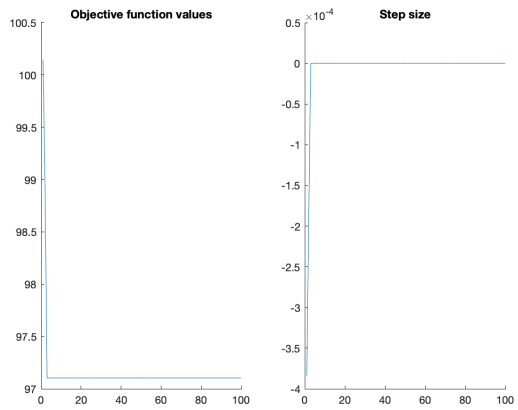
$Ax - b = 0$

$x \succeq 0$

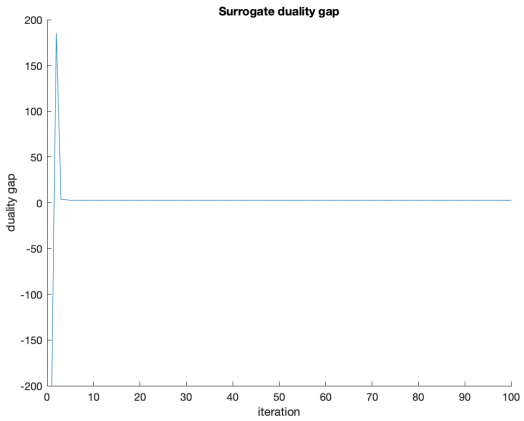
ii. By page 609, we have the form for the residual vector, and we can see that as $f(x) = -x$, since the constraint is $-x \preceq 0$. Then, we end up with $r_t(x, \lambda, \nu) =$ $-\begin{bmatrix} c - 1^T\lambda + A^T\nu \\ diag(\lambda)x - (1/t)\mathbf{1}_m \\ Ax - b \end{bmatrix}$
Then, to get the main form, we can see the left matrix on page 610, and can end up with the following equation:
$\begin{bmatrix} 0 & -I & A^T \\ diag(\lambda) & diag(x) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\nu \end{bmatrix} = -\begin{bmatrix} c - 1^T\lambda + A^T\nu \\ diag(\lambda)x - (1/t)\mathbf{1}_m \\ Ax - b \end{bmatrix}$ Analytically, if the $3x3$ matrix is H, and then $\Delta y_{pd} = [\Delta x \ \ \Delta\lambda \ \ \Delta\nu]^T$ and the right hand side is g, then our equation is of the form $Hy_{pd} = g$ and we solve for $y_{pd} = H^{-1}g$

iii. For the primal-dual interior point method, assuming we have calculated the search direction, we then, as explained in the book on page 613, do a linear back track with respect to the residual vector, so while $\|r(x^+, \lambda^+, \nu^+)\| > (1 - \alpha s)\|r(x, \lambda, \nu)\| : s = \beta * s$ where the initial $s = .99sup\{s \in [0, 1] | \lambda + s\Delta\lambda \succeq 0\}$

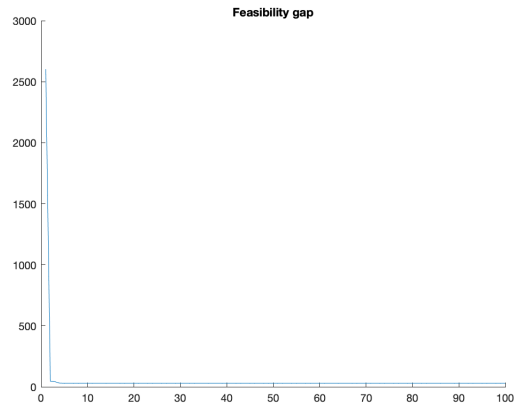
iv. The vast majority of my time on this exam has been spent on trying to implement this algorithm. At this point, I've missed the mistake in my derivation or bug in my code and have accepted that my answer isn't fully correct. I checked the optimal value using cvx and got a different value than these results, also KKT conditions don't appear to hold on my result. The code seems to work for a couple iterations then get stuck on a non-optimal value and stop. I'm not sure how to handle that, or what I'm missing that could cause it, but I'm tempted to blame issues with solving the matrix. At this point, I don't think I have enough time to figure out a fix. I understand the figures are supposed to somewhat resemble the ones in the book on 11.21. However, I am including what I have here:



v. Again, I know these aren't how the plots should look, and if I had more time or a better idea of what this bug in my code was, I would fix it. So, here is my surrogate duality gap graph:



Lastly, the the feasibility gap is on the next page, which does appear to go to close to 0:

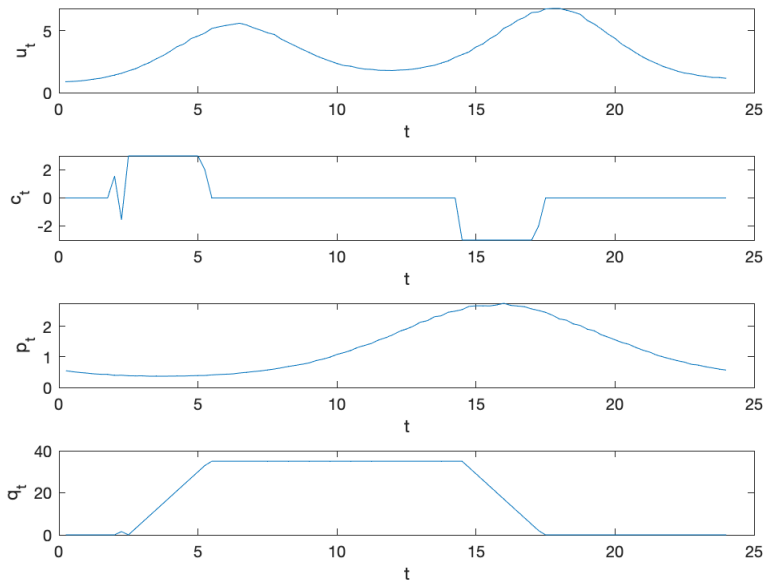


If I were more confident that this were accurate, which again, I don't think it is, we have quadratic convergence. Again, this point doesn't satisfy KKT conditions closely, especially the complementary slackness one, as $\eta = 2.5198$.

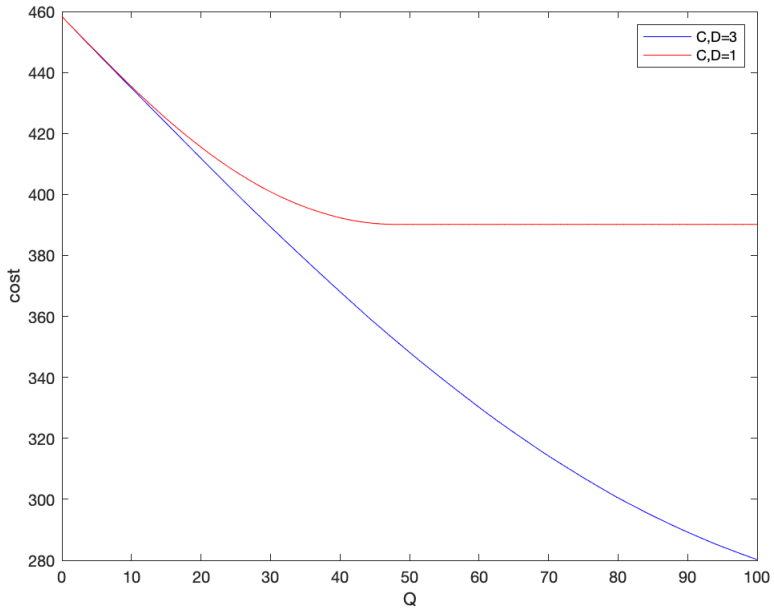
4. (a) This is a linear program. The objective function is $p^T(u+c)$. Then, combining the facts about the constraints the full problem is:

$$\text{minimize } p^T(u+c)$$
$$\text{such that } -D\mathbf{1} \preceq c \preceq C\mathbf{1}, \quad 0 \preceq q \preceq Q\mathbf{1}, \quad u + c \succeq 0$$
$$q_1 = q_T + c_T$$
$$q_{t+1} = q_t + c_t,\ t = 1, ...T-1$$

(b) Coded in MATLAB using the cvx package, which should be properly working (Apple doesn't like it's since not considered a verified developer, but the code did run). I solved the problem, and here is my completed graph. From top to bottom, the plots are of u, c, p, and q all versus t.



(c) For part c, I varied the values of Q between 1 and 100. I ran the code for $C, D = 1$ and $C, D = 3$. The plot is here:



We can see that for $C, D = 1$, the cost levels out as eventually the capacity constraint is large enough that it doesn't impact the other constraints. The end point should then be

the minimum cost. For the graph of $C, D = 3$, though we don't reach it here, it makes sense that that if we were to extend this graph to higher Q, then we would eventually realize it.

5. Question 6.

   (a) To do it without enumerating over all paths/node, for any node n, then we can define the recursive relation where $P_n = \underset{k \ inprev(n)}{max}(p_{nk}P_k)$ where $prev(n)$ is the set of all nodes that have paths leading to node n. p_{nk} is the probability of failure along that given path. Additionally, what we can do here is work with logarithms, and actually define $\log p_k = -a_k x_k$. Additionally, the probability of the smuggler being at the start node is 1, and $logP_1 = 0$. Using this, and letting $y_i = log(P_i)$:

$$\text{minimize } y_n$$
$$\text{such that } y_k = \underset{i \in prev(k)}{max} -a_{ik}x_{ik} + y_i \text{ for } k = 2.....n$$
$$y_1 = 0$$
$$\mathbf{1}^T x \leq B \text{ and } 0 \preceq x \preceq x^{max}$$

   Since we're multiplying probabilities, then any product of them will be non-increasing, so we can relax the first constraint into $y_k \geq \underset{i \in prev(k)}{max} -a_{ik}x_{ik} + y_i$.
   At this step, we can now incorporate the incidence matrix A. So $A^T z$ is a vector represents edges and paths going from one node to another. So, we can re-contextualize the first constraint in that terms and rewrite it as $A^T z \succeq -diag(\alpha)x$. Since $A^T z$ kind of represents the maximum chance. This gives us the final system of:

$$\text{minimize } y_n$$
$$\text{such that } -diag(\alpha)x \preceq A^T z$$
$$y_1 = 0$$
$$\mathbf{1}^T x \leq B \text{ and } 0 \preceq x \preceq x^{max}$$

   We can see that this is actually a linear program, as we have a linear objective function and linear constraints.

(b) Implemented using cvx in MATLAB. With non uniform cost, we get an optimal value of $P^{max*} = -3.13997$. Since this is the log of the probability, then $P^{max} = e^{-3.13997} = 0.043$ or 4.3%. If cost is then uniform, we lose the constraints relating to x^{max} and $1^T x \leq B$ and x is no longer a variable present in the problem. Running the problem under uniform cost gives the result $P^{max*} = -1.39812$ which we can then backtrack and get $P^{max} = e^{-1.39812} = 0.2471$ or a probability of 24.71%.

Code: I apologize in advance for the very messy code. All code listed here is what's in the zip file. My run time outputs are essentially graphs that I've included for each problem. I have them saved and can send the pictures if you would like. Everything was coded in MATLAB, so I just called variables from the workspace.

```
%Question 2 code:
%Load in and initialization
%Just comment in/out Newton's method/Gradient descents
load('P2_Parameters.mat', 'A')
load('P2_Parameters.mat', 'b')
alpha = 0.1;
beta = 0.5;
maxiters = 2000;
x = zeros(10,1);

m = 84; %84 or 142 depending on a/B, change accordingly
vals = zeros(1,m); %Fix number of trials for display purposes
grads = zeros(1,m);
steps = zeros(1,m);

 k = 1;
 while (norm(x + A'*(1./(b-A*x))) > 1e-5)
      f = 0.5*x'*x - sum(log(b-A*x));
```

```
    vals(k) = f;
    grad = x + A'*(1./(b-A*x));
    grads(k) = norm(grad);
    v = -grad; %Gradient descent
    fprime = grad'*v;
    t = 1;
    dx = -(A*v)./(b-A*x);

    %Line search, first verify that we are in the domain of the objective
    while (min(1+t*dx) < 0)
        t = beta*t;
    end

    %Once we're in the domain, we can do the backwards line search
    while (f + t*x'*v - sum(log(1+t*dx)) > f + t*alpha*fprime)
        t = beta*t;
    end
    steps(k) = t;
    x =  x+t*v;
    k = k + 1;
end

% sgtitle('Objective function vs iteration/step size \alpha = 0.03, \beta=0.3')
% subplot(1, 2, 1);
% title('Objective vs iteration number'); hold on
% plot(vals, 'b');
% xlabel('iteration');
% ylabel('Objective function');
```

```
% xlim([0 140]);
% subplot(1, 2, 2);
% title('Step size vs iteration number'); hold on
% plot(steps, 'b');
% xlabel('iteration');
% ylabel('Objective function');
% xlim([0 140]);

opt = 39.943;
test = vals - opt;
set(gca, 'YScale', 'log');
title('Norm of gradient vs iteration \alpha=0.1, \beta=0.5'); hold on
plot(grads, 'b');
ylabel('Norm of gradient in log scale');
xlabel('Iteration');
plot(test, 'r');
legend('\nabla f(x)', '(f-p^*)');
xlim([0,m-1]);

%Gradient descent above
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Newton's method below

% n = 9;
% load('P2_Parameters.mat', 'A')
% load('P2_Parameters.mat', 'b')
% vals = zeros(1,n); %From testing it takes 9 iterations either way
% grads = zeros(1,n);
```

```
% steps = zeros(1,n);
%
% x = zeros(10, 1);
% k = 1;
%
% dec = 1;
% decs = zeros(1,n);
% while (dec > 2e-12)
%     y = b-A*x;
%     f = x'*x - sum(log(y));
%     vals(k) = f;
%     grad = x + A'*(1./y); %Gradient and Hessian
%     hess = ones(10, 10) + A'*diag(1./(y.^2))*A;
%     v = -hess\grad;
%     dec = -grad'*v;
%     decs(k) = dec;
%     if (dec < 2e-12) %Break condition
%         break; end
%     fprime = grad'*v;
%     t = 1;
%     dx = -(A*v)./y;
%     dc = x'*v;
%
%    %Makes sure we are in the domain of the function
%     while (min(1+t*dx) < 0)
%         t = beta*t;
%     end
%
```

```
%     %Backwards line search in action
%     while (f + t*x'*v - sum(log(1+t*dx)) > f + t*alpha*fprime)
%           t = beta*t;
%     end
%     steps(k) = t;
%     x =  x+t*v;
%     k = k + 1;
% end

%All the plots included:
% sgtitle('Objective function vs iteration/step size \alpha = 0.03, \beta=0.3')
% subplot(1, 2, 1);
% title('Objective vs iteration number'); hold on
% plot(linspace(0, 9, 9), vals, 'b');
% xlabel('iteration');
% ylabel('Objective function');
% xlim([0 n]);
% subplot(1, 2, 2);
% title('Step size vs iteration number'); hold on
% plot(linspace(0, 9, 9), steps, 'b');
% xlabel('iteration');
% ylabel('Objective function');
% xlim([0 n]);

% opt = 39.9898;
%%test = log(abs(vals - opt + 1e-6));
% test = vals - opt;
% set(gca, 'YScale', 'log')
```

```
% title('Norm of gradient vs iteration \alpha=0.03, \beta=0.3'); hold on
% plot(linspace(0, 9, 9), decs, 'b');
% ylabel('Norm of gradient in log scale');
% xlabel('Iteration');
% plot(linspace(0, 9, 9), test, 'r');
% legend('\lambda^2', 'log(f-p^*)');
% xlim([0,n-1]);
% disp('x');

%Q3 code
load('P3_Parameters.mat', 'A');
load('P3_Parameters.mat', 'c');
load('P3_Parameters.mat', 'b');
%All parameters can be loaded in for file
max_iter = 10000;
eps = 10e-8;
eps_feas = 10e-8;
alpha = 0.01;
beta = 0.5;
mu = 10;
m = 50;
x = ones(200, 1);
l = ones(200, 1);
nu = zeros(m, 1);
vals = zeros(100, 1);
etas = zeros(100, 1);
steps = zeros(100, 1);
r_feas = zeros(100, 1);
iter = 1;
```

```
if iter > max_iter %If it really doesn't ever stop for some reason
    break; end
vals(iter, 1) = c'*x;
eta = -sum(l);
etas(iter, 1) = eta;

t = eta/mu*200;
b1 = [zeros(200, 200), -eye(200), A'; diag(l), diag(x), zeros(200,50); A, zeros(
b2 = [c - sum(l)+A'*nu; diag(l)*x - t*ones(200,1); b - A*x];
sol = -mldivide(b1,b2);

if (norm(A*x-b) < eps_feas && norm(c - sum(l)+A'*nu) < eps_feas && nu < eps)
    break;
end

dx = sol(1:200);
dl = sol(201:400);
dnu = sol(401:450);

s = .99*min(1, max(-dl./l));
while (min(x + s*dx) < 0)
    s = beta*s;
end
xtemp = x +s*dx; %Temporary updates
ltemp = l + s*dl;
nutemp = nu + s*dnu;
rup = [c - sum(ltemp) + A'*nu; diag(ltemp)*xtemp - t*ones(200,1); b - A*xtemp];
while(norm(rup) > (1-alpha*s)*norm(b2))
```

```
        s = s*beta;
        xtemp = x +s*dx;
        ltemp = l + s*dl;
        nutemp = nu + s*dnu;
        rup = [c - sum(ltemp) + A'*nu; diag(ltemp)*xtemp - t*ones(200,1); b - A*xte
    end
    steps(iter, 1) = s;
    x = xtemp;
    l = ltemp;
    nu = nutemp;
    r_feas(iter, 1) = sqrt(norm(c - sum(ltemp) + A'*nu)^2 + norm(A*x - b)^2);
    iter = iter + 1;
end

% subplot(1, 2, 1);
% title('Objective function values'); hold on
% plot(vals);
% subplot(1, 2, 2);
% title('Step size'); hold on
% plot(steps);
% title('Surrogate duality gap'); hold on
% plot(etas);
% xlabel('iteration');
% ylabel('duality gap');
title('Feasibility gap'); hold on
plot(r_feas);
xlabel('iteration');
ylabel('Feasibility gap');
```

```
%Q4 code
%Initialization and variable stuff
%Note I've commented out part a stuff for now,
randn('seed', 1);
T = 96; % 15 minute intervals in a 24 hour period
t = (1:T)';
p = exp(-cos((t-15)*2*pi/T)+0.01*randn(T,1));
u = 2*exp(-0.6*cos((t+40)*pi/T) -0.7*cos(t*4*pi/T)+0.01*randn(T,1));
Q = 35;
C = 3;
D = 3;

% Part a convex problem
% cvx_quiet(true);
% cvx_begin
%     variables q(T,1) c(T,1);
%     minimize(p'*(u+c));
%     subject to
%     c <= C;
%     c >= -D;
%     q <= Q;
%     q >= 0;
%     q(2:T) == q(1:T-1) + c(1:T-1);
%     q(1) == q(T) + c(T);
%     u+c >= 0;
% cvx_end

%Makes the four graphs displayed in my submission for 4a
```

```
% figure;
% ts = (1:T)/4;
% subplot(4,1,1);
% plot(ts,u); hold on
% xlabel('t');
% ylabel('u_t');
% subplot(4,1,2);
% plot(ts, c);
% xlabel('t');
% ylabel('c_t');
% subplot(4,1,3);
% plot(ts, p);
% ylabel('p_t');
% xlabel('t');
% subplot(4,1,4);
% plot(ts,q);
% ylabel('q_t');
% xlabel('t');
%To run part a code, just uncomment the code, and probably comment out the
%part below
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Variable initialization, I chose between 0 and 100 just because
Q = linspace(0, 100);

%Variables storing results of
Q_res = zeros(T, 100);
C_res = zeros(T, 100);
```

```
c_res = zeros(T, 1);
cvx_quiet(true);

%Loops through and does problem 100 times for C,D=3 case
for i = 1:100
    cvx_begin
        Q_curr = Q(i);
        variables q(T,1) c(T,1);
        minimize(p'*(u+c));
        c <= C; %Not sure what to use instead of <= but this works
        c >= -D;
        q <= Q_curr;
        q >= 0;
        q(2:T) == q(1:T-1) + c(1:T-1);
        q(1) == q(T) + c(T);
        u+c >= 0;
    cvx_end
    Q_res(:,i) = q; %Saves results just in case
    C_res(:,i) = c;
    c_res(i) = cvx_optval;
end

%Now does the case for C,D=1
C = 1;
D = 1;
Q_2 = linspace(0, 100);
c_res_2 = zeros(T, 1);
cvx_quiet(true);
```

```
for i = 1:100
    cvx_begin
        Q_curr = Q_2(i);
        variables q(T,1) c(T,1); minimize(p'*(u+c));
        c <= C;
        c >= -D;
        q <= Q_curr;
        q >= 0;
        q(2:T) == q(1:T-1) + c(1:T-1);
        q(1) == q(T) + c(T);
        u+c >= 0;
    cvx_end
    c_res_2(i) = cvx_optval; %Saves optimal value for graph
end

%Now that both have run, generate plots and compare
plot(Q, c_res, 'b'); hold on
plot(Q, c_res_2, 'r');
legend('C,D=3', 'C,D=1');
xlabel('Q');
ylabel('cost');

%Code for Q6
%Code from P6_Parameters.m for intialization
rand('state',0);
n=10;m=20;
edges=[[1 1 1 2 2 2 3 3 4 4 5 5 6 6 7 7 8 7 8 9]'...
    [2 3 4 6 3 4 5 6 6 7 8 7 7 8 8 9 9 10 10 10]'];
A=zeros(n,m);
```

```
for j=1:size(edges,1)
    A(edges(j,1),j)=-1;A(edges(j,2),j)=1;
end
a=2*rand(m,1);
x_max = 1+rand(m,1);
B=m/2;
%Just comment out the one you don't want to run for simplicity

%Part a w/o uniform cost assumption
% cvx_begin
%     variables x(m) z(n);
%     minimize(z(n));
%     z(1) == 0;
%     A'*z >= -diag(a)*x;
%     x >= 0;
%     x <= x_max;
%     sum(x) <= B;
% cvx_end

%With uniform cost, outlined in answer where simplification comes from
x = B/m*ones(m,1);
cvx_begin
    variables z2(n);
    minimize(z2(n));
    z2(1) == 0;
    A'*z2 >= -diag(a)*x;
cvx_end
```