EE364a Homework 8 solutions

9.30 *Gradient and Newton methods.* Consider the unconstrained problem

$$\text{minimize} \quad f(x) = -\sum_{i=1}^m \log(1 - a_i^T x) - \sum_{i=1}^n \log(1 - x_i^2),$$

with variable $x \in \mathbf{R}^n$, and $\mathbf{dom}\, f = \{x \mid a_i^T x < 1,\ i = 1, \ldots, m,\ |x_i| < 1,\ i = 1, \ldots, n\}$. This is the problem of computing the analytic center of the set of linear inequalities

$$a_i^T x \leq 1, \quad i = 1, \ldots, m, \qquad |x_i| \leq 1, \quad i = 1, \ldots, n.$$

Note that we can choose $x^{(0)} = 0$ as our initial point. You can generate instances of this problem by choosing a_i from some distribution on $\mathbf{R}^n$.

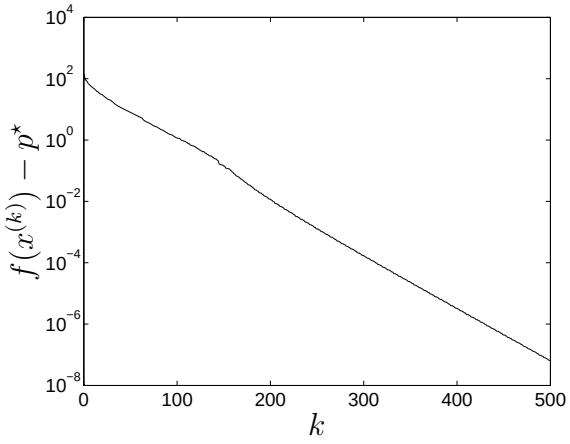
(a) Use the gradient method to solve the problem, using reasonable choices for the backtracking parameters, and a stopping criterion of the form $\|\nabla f(x)\|_2 \leq \eta$. Plot the objective function and step length versus iteration number. (Once you have determined $p^\star$ to high accuracy, you can also plot $f - p^\star$ versus iteration.) Experiment with the backtracking parameters α and β to see their effect on the total number of iterations required. Carry these experiments out for several instances of the problem, of different sizes.

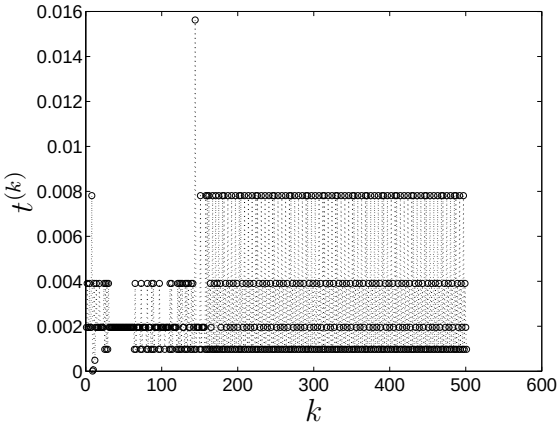
(b) Repeat using Newton's method, with stopping criterion based on the Newton decrement λ^2. Look for quadratic convergence. You do not have to use an efficient method to compute the Newton step, as in exercise 9.27; you can use a general purpose dense solver, although it is better to use one that is based on a Cholesky factorization.

Hint. Use the chain rule to find expressions for $\nabla f(x)$ and $\nabla^2 f(x)$.

Solution.

(a) *Gradient method.* The figures show the function values and step lengths versus iteration number for an example with $m = 200$, $n = 100$. We used $\alpha = 0.01$, $\beta = 0.5$, and exit condition $\|\nabla f(x^{(k)})\|_2 \leq 10^{-3}$.

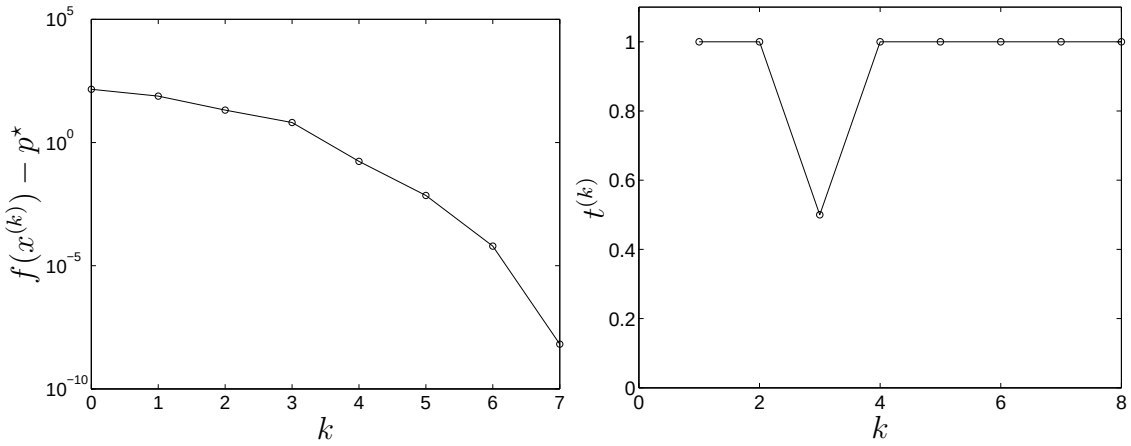




The following is a Matlab implementation.

```
ALPHA = 0.01;
BETA = 0.5;
MAXITERS = 1000;
GRADTOL = 1e-3;

x = zeros(n,1);
for iter = 1:MAXITERS
   val = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
   grad =  A'*(1./(1-A*x)) - 1./(1+x) + 1./(1-x);
   if norm(grad) < GRADTOL, break; end;
   v = -grad;
   fprime = grad'*v;
   t = 1;  while ((max(A*(x+t*v)) >= 1) | (max(abs(x+t*v)) >= 1)),
      t = BETA*t;
   end;
   while ( -sum(log(1-A*(x+t*v))) - sum(log(1-(x+t*v).^2)) >  ...
           val + ALPHA*t*fprime )
      t = BETA*t;
   end;
   x = x+t*v;
end;
```

(b) *Newton method.* The figures show the function values and step lengths versus iteration number for the same example. We used $\alpha = 0.01$, $\beta = 0.5$, and exit condition $\lambda(x^{(k)})^2 \leq 10^{-8}$.



The following is a Matlab implementation.

```
ALPHA = 0.01;
BETA = 0.5;
MAXITERS = 1000;
NTTOL  = 1e-8;

x = zeros(n,1);
for iter = 1:MAXITERS
   val = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
   d = 1./(1-A*x);
   grad =  A'*d - 1./(1+x) + 1./(1-x);
   hess =  A'*diag(d.^2)*A + diag(1./(1+x).^2 + 1./(1-x).^2);
   v = -hess\grad;
   fprime = grad'*v;
   if abs(fprime) < NTTOL, break; end;
   t = 1;  while ((max(A*(x+t*v)) >= 1) | (max(abs(x+t*v)) >= 1)),
      t = BETA*t;
   end;
   while ( -sum(log(1-A*(x+t*v))) - sum(log(1-(x+t*v).^2)) >  ...
           val + ALPHA*t*fprime )
      t = BETA*t;
   end;
   x = x+t*v;
end;
```

10.8 *Infeasible start Newton method and initially satisfied equality constraints.* Suppose we use the infeasible start Newton method to minimize $f(x)$ subject to $a_i^T x = b_i$, $i = 1, \ldots, p$.

(a) Suppose the initial point $x^{(0)}$ satisfies the linear equality $a_i^T x = b_i$. Show that the linear equality will remain satisfied for future iterates, *i.e.*, if $a_i^T x^{(k)} = b_i$ for all k.

(b) Suppose that one of the equality constraints becomes satisfied at iteration k, *i.e.*, we have $a_i^T x^{(k-1)} \neq b_i$, $a_i^T x^{(k)} = b_i$. Show that at iteration k, *all* the equality constraints are satisfied.

Solution.
Follows easily from

$$r^{(k)} = \left(\prod_{i=0}^{k-1} (1 - t^{(i)}) \right) r^{(0)}.$$

A16.5 *Minimum energy processor speed scheduling.* A single processor can adjust its speed in each of T time periods, labeled $1, \ldots, T$. Its speed in period t will be denoted s_t, $t = 1, \ldots, T$. The speeds must lie between given (positive) minimum and maximum values, $S^{\min}$ and $S^{\max}$, respectively, and must satisfy a slew-rate limit, $|s_{t+1} - s_t| \leq R$, $t = 1, \ldots, T-1$. (That is, R is the maximum allowed period-to-period change in speed.) The energy consumed by the processor in period t is given by $\phi(s_t)$, where $\phi : \mathbf{R} \to \mathbf{R}$ is increasing and convex. The total energy consumed over all the periods is $E = \sum_{t=1}^{T} \phi(s_t)$.

The processor must handle n jobs, labeled $1, \ldots, n$. Each job has an availability time $A_i \in \{1, \ldots, T\}$, and a deadline $D_i \in \{1, \ldots, T\}$, with $D_i \geq A_i$. The processor cannot start work on job i until period $t = A_i$, and must complete the job by the end of period D_i. Job i involves a (nonnegative) total work W_i. You can assume that in each time period, there is at least one job available, *i.e.*, for each t, there is at least one i with $A_i \leq t$ and $D_i \geq t$.

In period t, the processor allocates its effort across the n jobs as θ_t, where $\mathbf{1}^T \theta_t = 1$, $\theta_t \succeq 0$. Here θ_{ti} (the ith component of θ_t) gives the fraction of the processor effort devoted to job i in period t. Respecting the availability and deadline constraints requires that $\theta_{ti} = 0$ for $t < A_i$ or $t > D_i$. To complete the jobs we must have

$$\sum_{t=A_i}^{D_i} \theta_{ti} s_t \geq W_i, \quad i = 1, \ldots, n.$$

(a) Formulate the problem of choosing the speeds $s_1, \ldots, s_T$, and the allocations $\theta_1, \ldots, \theta_T$, in order to minimize the total energy E, as a convex optimization problem. The problem data are $S^{\min}$, $S^{\max}$, R, ϕ, and the job data, A_i, D_i, W_i, $i = 1, \ldots, n$. Be sure to justify any change of variables, or introduction of new variables, that you use in your formulation.

(b) Carry out your method on the problem instance described in `proc_sched_data.m`, with quadratic energy function $\phi(s_t) = \alpha + \beta s_t + \gamma s_t^2$. (The parameters α, β, and γ are given in the data file.) Executing this file will also give a plot showing the availability times and deadlines for the jobs.

Give the energy obtained by your speed profile and allocations. Plot these using the command `bar((s*ones(1,n)).*theta,1,'stacked')`, where s is the $T \times 1$ vector of speeds, and θ is the $T \times n$ matrix of allocations with components θ_{ti}. This will show, at each time period, how much effective speed is allocated to each job. The top of the plot will show the speed s_t. (You don't need to turn in a color version of this plot; B&W is fine.)

Solution. The trick is to work with the variables $x_{ti} = \theta_{ti} s_t$, which must be nonnegative. Let $X \in \mathbf{R}^{T \times n}$ denote the matrix with components x_{ti}. The job completion constraint can be expressed as $X^T \mathbf{1} \succeq W$. The availability and deadline constraints can be expressed as $x_{ti} = 0$ for $t < A_i$ or $t > D_i$, which are linear constraints. The speed can be expressed as $s = X\mathbf{1}$, a linear function of our variable X. The objective E is clearly a convex function of s (and so, also of X).

Our convex optimization problem is

$$\begin{array}{ll} \text{minimize} & E = \sum_{t=1}^{T} \phi(s_t) \\ \text{subject to} & S^{\min} \preceq s \preceq S^{\max}, \quad s = X\mathbf{1}, \quad X^T\mathbf{1} \succeq W, \quad X \succeq 0 \\ & |s_{t+1} - s_t| \leq R, \quad t = 1, \ldots, T-1 \\ & X_{ti} = 0, \quad t = 1, \ldots, A_i - 1, \quad i = 1, \ldots, n \\ & X_{ti} = 0, \quad t = D_i + 1, \ldots, T, \quad i = 1, \ldots, n \end{array}$$

with variables $s \in \mathbf{R}^T$ and $X \in \mathbf{R}^{T \times n}$. All generalized inequalities here, including $X \succeq 0$, are elementwise nonnegativity. Evidently this is a convex problem.

Once we find an optimal $X^\star$ and $s^\star$, we can recover $\theta_t^\star$ as

$$\theta_{ti}^\star = (1/s_t^\star) x_{ti}^\star, \quad t = 1, \ldots, T, \quad i = 1, \ldots, n.$$

For our data, the availability and deadlines of jobs are plotted in figure 1. The job duration lines show a start time at the beginning of the available time period and a termination time at the very end of the deadline time period. Thus, the length of the line shows the actual duration within which the job is allowed to be completed.

The code that solves the problem is as follows.

```
proc_sched_data;

cvx_begin
    variable X(T,n)
    X >= 0;
    s = sum(X')';
    minimize(sum(alpha+beta*s+gamma*square(s)))
    s >= Smin;
    s <= Smax;
```

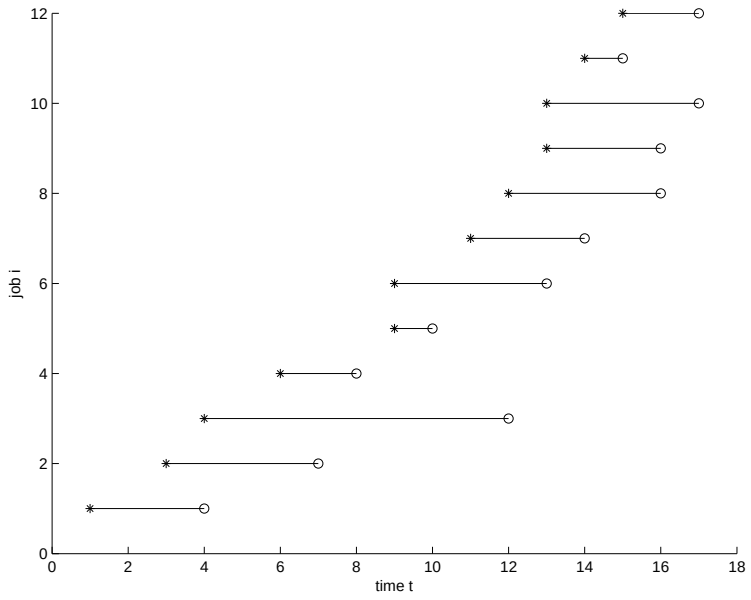


Figure 1 Job availability diagram. Stars indicate the time in which each job becomes available. Open circles indicate the required end of the job, which is at the end of the deadline time period.

```
    abs(s(2:end)-s(1:end-1))<=R; % slew rate constraint

    % start/stop constraints
    for i=1:n
        for t=1:A(i)-1
            X(t,i)==0;
        end
        for t=D(i)+1:T
            X(t,i)==0;
        end
    end

    sum(X)>=W';

cvx_end
theta = X./(s*ones(1,n));

figure;
bar((s*ones(1,n)).*theta,1,'stacked');
xlabel('t');
```

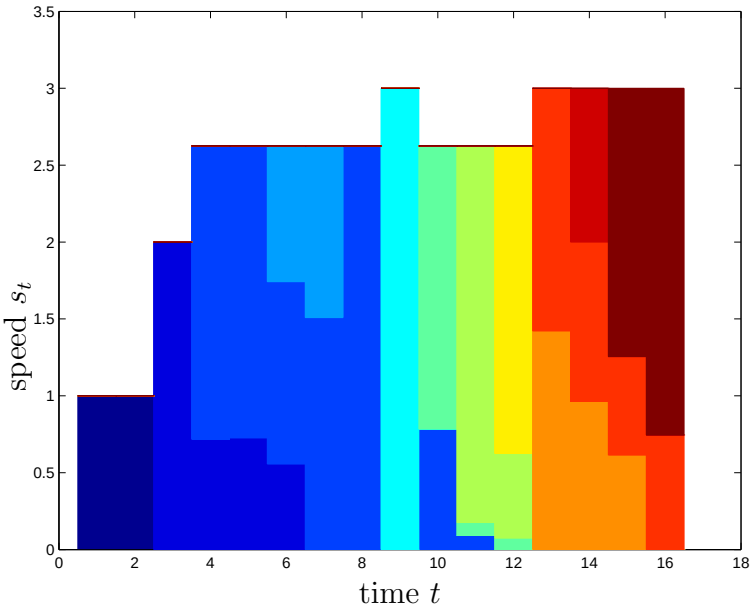


Figure 2 Optimal speed allocation profile. The top of the plot gives the processor speed. The colored regions indicate the portion of the speed allocated to a particular job at each time.

```
ylabel('st');
```

The optimal total energy is $E = 162.125$, which is obtained by allocating speeds according to the plot shown in figure 2.

3.27 *Affine policy.* We consider a family of LPs, parametrized by the random variable u, which is uniformly distributed on $\mathcal{U} = [-1, 1]^p$,

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b(u), \end{array}$$

where $x \in \mathbf{R}^n$, $A \in \mathbf{R}^{m\times n}$, and $b(u) = b_0 + Bu \in \mathbf{R}^m$ is an affine function of u. You can think of u_i as representing a deviation of the ith parameter from its nominal value. The parameters might represent (deviations in) levels of resources available, or other varying limits.

The problem is to be solved many times; in each time, the value of u (*i.e.*, a sample) is given, and then the decision variable x is chosen. The mapping from u into the decision variable $x(u)$ is called the *policy*, since it gives the decision variable value for each value of u. When enough time and computing hardware is available, we can simply solve the LP for each new value of u; this is an optimal policy, which we denote $x^\star(u)$.

In some applications, however, the decision $x(u)$ must be made very quickly, so solving the LP is not an option. Instead we seek a suboptimal policy, which is affine: $x^{\mathrm{aff}}(u) = x_0 + Ku$, where x_0 is called the *nominal decision* and $K \in \mathbf{R}^{n\times p}$ is called the *feedback gain matrix*. (Roughly speaking, x_0 is our guess of x before the value of u has been revealed; Ku is our modification of this guess, once we know u.) We determine the policy (*i.e.*, suitable values for x_0 and K) ahead of time; we can then evaluate the policy (that is, find $x^{\mathrm{aff}}(u)$ given u) very quickly, by matrix multiplication and addition.

We will choose x_0 and K in order to minimize the expected value of the objective, while insisting that for any value of u, feasibility is maintained:

$$\begin{array}{ll} \text{minimize} & \mathbf{E}\, c^T x^{\mathrm{aff}}(u) \\ \text{subject to} & Ax^{\mathrm{aff}}(u) \preceq b(u) \quad \forall u \in \mathcal{U}. \end{array}$$

The variables here are x_0 and K. The expectation in the objective is over u, and the constraint requires that $Ax^{\mathrm{aff}}(u) \preceq b(u)$ hold almost surely.

(a) Explain how to find optimal values of x_0 and K by solving a standard explicit convex optimization problem (*i.e.*, one that does not involve an expectation or an infinite number of constraints, as the one above does.) The numbers of variables or constraints in your formulation should not grow exponentially with the problem dimensions n, p, or m.

(b) Carry out your method on the data given in `affine_pol_data.m`. To evaluate your affine policy, generate 100 independent samples of u, and for each value, compute the objective value of the affine policy, $c^T x^{\mathrm{aff}}(u)$, and of the optimal policy, $c^T x^\star(u)$. Scatter plot the objective value of the affine policy (y-axis) versus the objective value of the optimal policy (x-axis), and include the line $y = x$ on the plot. Report the average values of $c^T x^{\mathrm{aff}}(u)$ and $c^T x^\star(u)$ over your samples. (These are estimates of $\mathbf{E}\, c^T x^{\mathrm{aff}}(u)$ and $\mathbf{E}\, c^T x^\star(u)$. The first number, by the way, can be found exactly.)

Solution. Let's start with the objective. We compute the expected value of the affine policy as

$$\mathbf{E}\, c^T(x_0 + Ku) = c^T x_0 + (K^T c)^T\, \mathbf{E}\, u = c^T x_0.$$

Now let's look at the constraints, which we write out in terms of its entries:

$$(Ax_0)_i + \sup_{u\in\mathcal{U}}((AK - B)u)_i \leq (b_0)_i, \quad i = 1, \ldots, m.$$

This turns into the explicit constraint

$$(Ax_0)_i + \|(AK - B)_i\|_1 \leq (b_0)_i, \quad i = 1, \ldots, m.$$

Here $(AK - B)_i$ is the ith row of the matrix $A - BK$.

So we can find an optimal affine policy by solving the problem (which can be transformed to an LP)

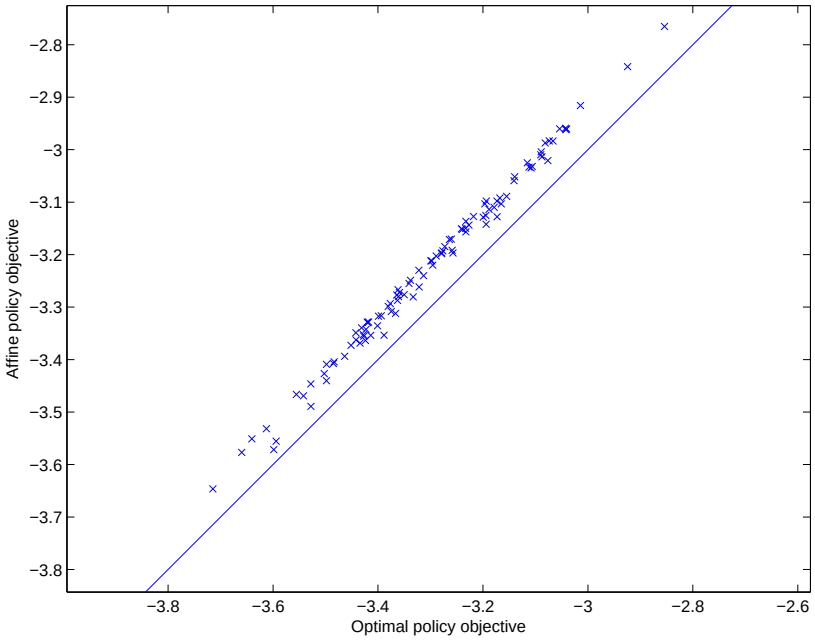
$$\begin{array}{ll} \text{minimize} & c^T x_0 \\ \text{subject to} & (Ax_0)_i + \|(AK-B)_i\|_1 \leq (b_0)_i, \quad i = 1, \ldots, m, \end{array}$$

with variables x_0 and K.

The code below implements this.

```
% Affine policy.
affine_pol_data;

% compute affine policy
cvx_begin
    variables x0(n) K(n,p)
    minimize (c'*x0)
    subject to
        A*x0+norms(A*K-B,1,2) <= b0
cvx_end

% compare 100 samples
aff_obj = []; opt_obj = [];
for i=1:100
    u = 2*rand(p,1)-1;

    cvx_begin quiet
        variable x_opt(n)
        minimize (c'*x_opt)
        subject to
            A*x_opt <= b0+B*u
    cvx_end

    aff_obj(i) = c'*(x0+K*u);
    opt_obj(i) = cvx_optval;
end

figure;
plot(opt_obj,aff_obj,'x'); hold on;
axis equal
plot(xlim, xlim);
xlabel('Optimal policy objective')
ylabel('Affine policy objective')
```

```
print -depsc affine_pol.eps
```



The (exact) expected cost of the affine policy is -3.219, the mean objective of the affine policy is -3.223 for the sample, and the mean objective of the optimal policy is -3.301 for the sample. The plot shows that the affine policy produces points that are suboptinmal, but not by much.