

EE364a Homework 7 solutions

9.25 *Smoothed fit to given data.* Consider the problem

$$\text{minimize } f(x) = \sum_{i=1}^n \psi(x_i - y_i) + \lambda \sum_{i=1}^{n-1} (x_{i+1} - x_i)^2$$

where $\lambda > 0$ is smoothing parameter, ψ is a convex penalty function, and $x \in \mathbf{R}^n$ is the variable. We can interpret x as a smoothed fit to the vector y .

- (a) What is the structure in the Hessian of f ?
- (b) Extend to the problem of making a smooth fit to two-dimensional data, *i.e.*, minimizing the function

$$\sum_{i,j=1}^n \psi(x_{ij} - y_{ij}) + \lambda \left(\sum_{i=1}^{n-1} \sum_{j=1}^n (x_{i+1,j} - x_{ij})^2 + \sum_{i=1}^n \sum_{j=1}^{n-1} (x_{i,j+1} - x_{ij})^2 \right),$$

with variable $X \in \mathbf{R}^{n \times n}$, where $Y \in \mathbf{R}^{n \times n}$ and $\lambda > 0$ are given.

Solution.

- (a) Tridiagonal.
- (b) Block-tridiagonal if we store the elements of X columnwise. The blocks have size $n \times n$. The diagonal blocks are tridiagonal. The blocks on the first sub-diagonal are diagonal.

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

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

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

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

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 .

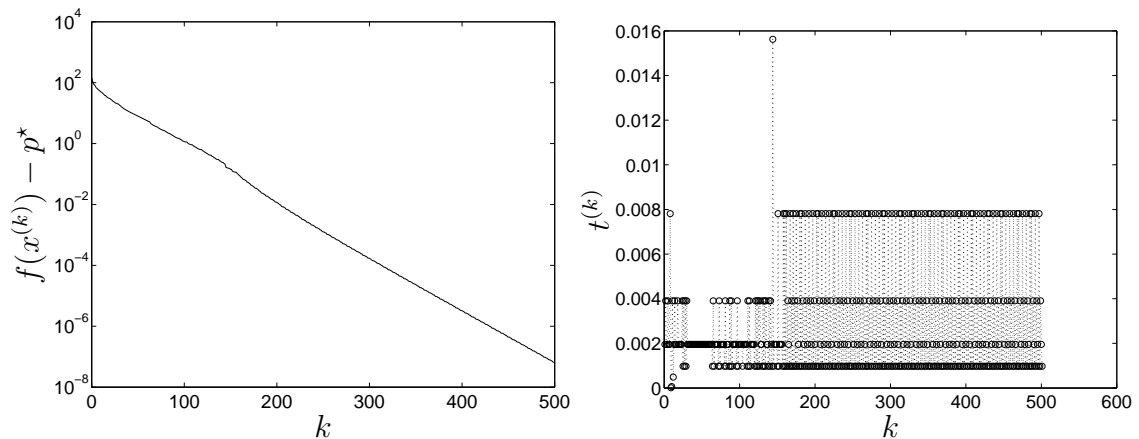
- (a) Use the gradient method to solve the problem, using reasonable choices for the backtracking parameters, and a stopping criterion of the form $\|\nabla f(x)\|_2 \leq \eta$. Plot the objective function and step length versus iteration number. (Once you have determined p^* to high accuracy, you can also plot $f - p^*$ 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.

```

randn('state',1);
m=200;
n=100;

ALPHA = 0.01;
BETA = 0.5;
MAXITERS = 1000;
NTTOL = 1e-8;
GRADTOL = 1e-3;

% generate random problem
A = randn(m,n);

% gradient method

vals = []; steps = [];

```

```

x = zeros(n,1);
for iter = 1:MAXITERS

    val = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
    vals = [vals, val];
    d = 1./(1-A*x);
    grad = A'*d - 1./(1+x) + 1./(1-x);
    v = -grad;

    fprime = grad'*v;
    norm(grad)
    if norm(grad) < GRADTOL, 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;
    steps = [steps,t];

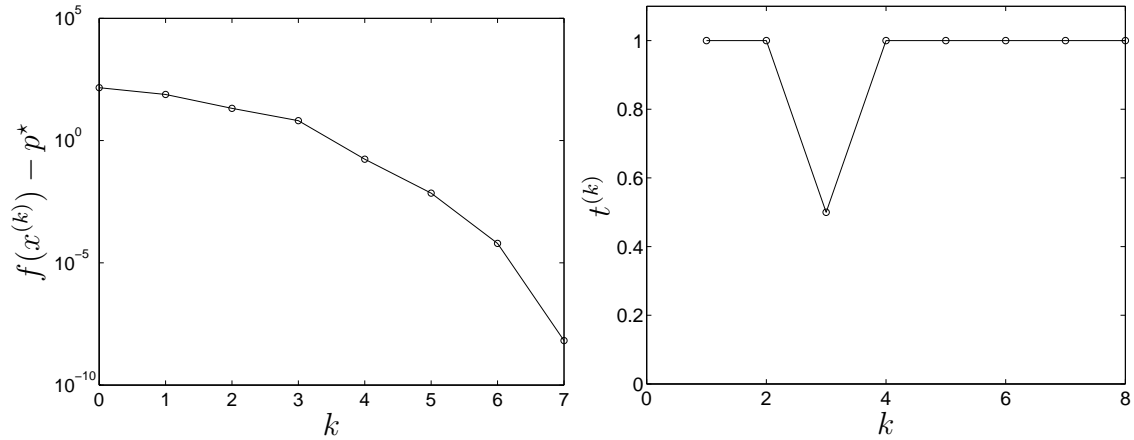
end;

figure(1)
semilogy([0:(length(vals)-2)], vals(1:length(vals)-1)-optval, '-');
xlabel('x'); ylabel('z');

figure(2)
plot([1:length(steps)], steps, ':',[1:length(steps)], steps, 'o');
xlabel('x'); ylabel('z');

```

- (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.

```
% Newton method

vals = []; steps = [];

x = zeros(n,1);
for iter = 1:MAXITERS

    val = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
    vals = [vals, val];
    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;
    steps = [steps,t];
end;
```

```

end;
optval = vals(length(vals));

figure(3)
semilogy([0:(length(vals)-2)], vals(1:length(vals)-1)-optval, '- ', ...
          [0:(length(vals)-2)], vals(1:length(vals)-1)-optval, 'o');
xlabel('x'); ylabel('z');

figure(4)
plot([1:length(steps)], steps, '- ', [1:length(steps)], steps, 'o');
axis([0, length(steps), 0, 1.1]);
xlabel('x'); ylabel('z');

```

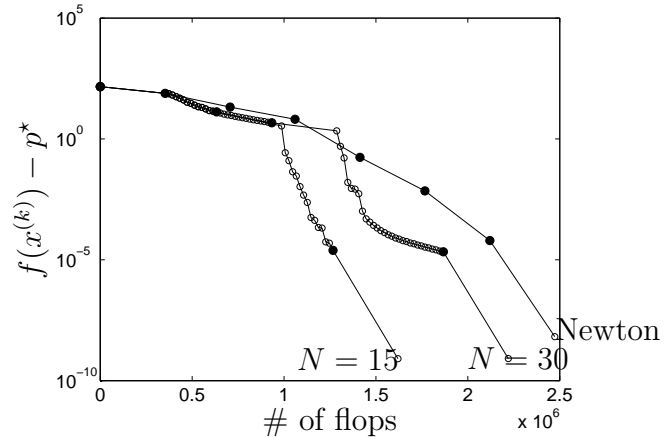
9.31 *Some approximate Newton methods.* The cost of Newton's method is dominated by the cost of evaluating the Hessian $\nabla^2 f(x)$ and the cost of solving the Newton system. For large problems, it is sometimes useful to replace the Hessian by a positive definite approximation that makes it easier to form and solve for the search step. In this problem we explore some common examples of this idea.

For each of the approximate Newton methods described below, test the method on some instances of the analytic centering problem described in exercise 9.30, and compare the results to those obtained using the Newton method and gradient method.

- (a) *Re-using the Hessian.* We evaluate and factor the Hessian only every N iterations, where $N > 1$, and use the search step $\Delta x = -H^{-1}\nabla f(x)$, where H is the last Hessian evaluated. (We need to evaluate and factor the Hessian once every N steps; for the other steps, we compute the search direction using back and forward substitution.)
- (b) *Diagonal approximation.* We replace the Hessian by its diagonal, so we only have to evaluate the n second derivatives $\partial^2 f(x)/\partial x_i^2$, and computing the search step is very easy.

Solution.

- (a) The figure shows the function value versus approximate total number of flops required (for the same example as in the solution of exercise 9.30), for $N = 1$ (*i.e.*, Newton's method), $N = 15$, and $N = 30$.



We see that the speed of convergence is increased using the method of using a factorized Hessian for several steps, as measured by true effort (*i.e.*, number of flops required). Of course in terms of iterations, the method is worse than the basic Newton method.

The following is a Matlab implementation.

```

randn('state',1);
m=200;
n=100;

ALPHA    = 0.01;
BETA     = 0.5;
MAXITERS = 1000;
NTTOL    = 1e-9;
GRADTOL  = 1e-3;

% generate random problem
A = randn(m,n);

% Newton method with periodically updated Hessian

for N = [1,15,30]; % re-compute Hessian every N iterations

    vals = [];
    flops = [];
    flop = 0;
    x = zeros(n,1);
    for iter = 1:MAXITERS

        val = -sum(log(1-A*x))-sum(log(1+x))-sum(log(1-x));
        vals = [vals,val];

```

```

flops = [flops,flop];

d = 1./(1-A*x);
grad = A'*d-1./(1+x)+1./(1-x);

if (rem(iter-1,N) == 0)
    H = A'*diag(d.^2)*A+diag(1./(1+x).^2+1./(1-x).^2);
    L = chol(H,'lower');
    flop = (1/3)*n^3; % add flop for Cholesky factorization
else
    flop = 0;
end
v = -L'\(L\grad);
flop = flop+2*n^2; % add flop for fwd/bwd substitution

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

if (N==1), optval = vals(length(vals)); end

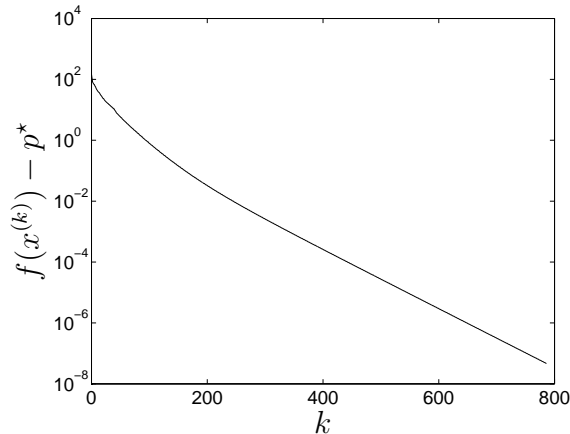
figure(1)
cflops = cumsum(flops(1:end-1));
perror = vals(1:end-1)-optval;
semilogy(cflops,perror,'-',cflops,perror,'o');
hold on;
semilogy(cflops(1:N:end-1),perror(1:N:end-1),...
    'mo','MarkerEdgeColor','k','MarkerFaceColor','b','MarkerSize',8);
text(cflops(end),perror(end),['N',num2str(N)]);
hold on;

end

```

```
xlabel('x'); ylabel('z');
```

- (b) The figure shows the function value versus iteration number (for the same example as in the solution of exercise 9.30), for a diagonal approximation of the Hessian. The experiment shows that the algorithm converges very much like the gradient method.



The following is a Matlab implementation.

```
% Newton method with diagonal approximation of Hessian

vals = [];
x = zeros(n,1);
for iter = 1:MAXITERS

    val = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
    vals = [vals, val];
    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);
    H = diag(diag(hess));
    norm(grad)
    if norm(grad) < GRADTOL, break; end;
    v = -H\grad; fprime = grad'*v;

    if (fprime > 0), keyboard; 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

figure(2)
semilogy([0:(length(vals)-2)], vals(1:length(vals)-1)-optval, '-');
xlabel('x'); ylabel('z');

```

10.1 *Nonsingularity of the KKT matrix.* Consider the KKT matrix

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix},$$

where $P \in \mathbf{S}_+^n$, $A \in \mathbf{R}^{p \times n}$, and $\mathbf{rank} A = p < n$.

- (a) Show that each of the following statements is equivalent to nonsingularity of the KKT matrix.
- $\mathcal{N}(P) \cap \mathcal{N}(A) = \{0\}$.
 - $Ax = 0, x \neq 0 \implies x^T Px > 0$.
 - $F^T P F \succ 0$, where $F \in \mathbf{R}^{n \times (n-p)}$ is a matrix for which $\mathcal{R}(F) = \mathcal{N}(A)$.
 - $P + A^T Q A \succ 0$ for some $Q \succeq 0$.
- (b) Show that if the KKT matrix is nonsingular, then it has exactly n positive and p negative eigenvalues.

Solution.

- (a) • *Conditions 1 and 2.* If $x \in \mathcal{N}(A) \cap \mathcal{N}(P)$, $x \neq 0$, then $Ax = 0$, $x \neq 0$, but $x^T Px = 0$, contradicting the second statement. Conversely, suppose the second statement fails to hold, *i.e.*, there is an x with $Ax = 0$, $x \neq 0$, but $x^T Px = 0$. Since $P \succeq 0$, we conclude $Px = 0$, *i.e.*, $x \in \mathcal{N}(P)$, which contradicts the first statement.
- *Conditions 2 and 3.* If $Ax = 0$, $x \neq 0$, then x must have the form $x = Fz$, where $z \neq 0$ because $\mathbf{rank}(F) = n-p$. Then we have $x^T Px = z^T F^T P F z > 0$.
- *Conditions 2 and 4.* If the second condition holds then

$$x^T (P + A^T A)x = x^T Px + \|Ax\|_2^2 > 0$$

for all nonzero x , so the last statement holds with $Q = I$.
If the last statement holds for some $Q \succeq 0$ then

$$x^T (P + A^T Q A)x = x^T Px + x^T A^T Q A x > 0$$

for all nonzero x . Therefore if $Ax = 0$ and $x \neq 0$, we must have $x^T Px > 0$.

Now let us show that the four statements are equivalent to nonsingularity of the KKT matrix. First suppose that x satisfies $Ax = 0$, $Px = 0$, and $x \neq 0$. Then

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = 0,$$

which shows that the KKT matrix is singular.

Now suppose the KKT matrix is singular, *i.e.*, there are x, z , not both zero, such that

$$\begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} = 0.$$

This means that $Px + A^T z = 0$ and $Ax = 0$, so multiplying the first equation on the left by x^T , we find $x^T Px + x^T A^T z = 0$. Using $Ax = 0$, this reduces to $x^T Px = 0$, so we have $Px = 0$ (using $P \succeq 0$). This contradicts (a), unless $x = 0$. In this case, we must have $z \neq 0$. But then $A^T z = 0$ contradicts $\mathbf{rank} A = p$.

Alternative solution.

We claim the following statements are equivalent:

1. The KKT matrix is nonsingular.
 2. $Ax = 0, x \neq 0 \implies x^T Px > 0$.
 3. $\mathcal{N}(P) \cap \mathcal{N}(A) = \{0\}$.
 4. $F^T P F \succ 0$, where $F \in \mathbf{R}^{n \times (n-p)}$ is a matrix for which $\mathcal{R}(F) = \mathcal{N}(A)$.
 5. $P + A^T Q A \succ 0$ for some $Q \succeq 0$.
- $1 \implies 2$: Suppose statement 2 is not true, then there exists an $x \neq 0$ such that $Ax = 0$ and $x^T Px = 0$. Since P is positive semidefinite this implies $Px = 0$, so the vector $\begin{bmatrix} x \\ 0 \end{bmatrix}$ is in the nullspace of the KKT matrix, which contradicts statement 1.
 - $2 \implies 3$: Suppose statement 3 is not true, then there exists an $x \neq 0$ such that $Px = 0$ and $Ax = 0$. This implies $Ax = 0$ and $x^T Px = 0$, which contradicts statement 2.
 - $3 \implies 4$: Suppose statement 4 is not true, then we can find $z \neq 0$ such that $z^T F^T P F z = 0$. Because $\mathbf{rank}(F) = n - p$ (*i.e.*, F is full rank), so $x = Fz$ satisfies $x \neq 0$, $Ax = 0$, and $Px = 0$, which contradicts statement 3.
 - $4 \implies 5$: Suppose statement 5 is not true, then for any $Q \succeq 0$, we can find an $x \neq 0$ such that $x^T (P + A^T Q A)x = 0$. This implies that $x^T Px + x^T A^T Q A x = 0$, but since both terms in the sum are nonnegative we must have $x^T Px = 0$ and $x^T A^T Q A x = 0$. Since this is true for any $Q \succeq 0$, it must be true for any $Q \succ 0$. Thus, we can find an $x \neq 0$ for which $x^T Px = 0$ and $Ax = 0$. Since $Ax = 0$ we can write $x = Fz$ for some $z \neq 0$. This implies $z^T F^T P F z = 0$, contradicting statement 4.

- 5 \Rightarrow 1: Suppose statement 1 is not true, then we can find a vector $\begin{bmatrix} x \\ y \end{bmatrix}$ that satisfies $Px + A^T y = 0$ and $Ax = 0$. This implies that $x^T Px + x^T A^T Q Ax = x^T Px = -x^T A^T y = -(Ax)^T y = 0$ for any Q , which contradicts statement 5, unless $x = 0$. But $x = 0$ would imply $y \neq 0$ and $A^T y = 0$, which contradicts $\text{rank } A = p$.

- (b) From part (a), $P + A^T A \succ 0$. Therefore there exists a nonsingular matrix $R \in \mathbf{R}^{n \times n}$ such that

$$R^T(P + A^T A)R = I.$$

Let $AR = U\Sigma V_1^T$ be the singular value decomposition of AR , with $U \in \mathbf{R}^{p \times p}$, $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbf{R}^{p \times p}$ and $V_1 \in \mathbf{R}^{n \times p}$. Let $V_2 \in \mathbf{R}^{n \times (n-p)}$ be such that

$$V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$$

is orthogonal, and define

$$S = \begin{bmatrix} \Sigma & 0 \end{bmatrix} \in \mathbf{R}^{p \times n}.$$

We have $AR = USV^T$, so

$$V^T R^T (P + A^T A) R V = V^T R^T P R V + S^T S = I.$$

Therefore $V^T R^T P R V = I - S^T S$ is diagonal. We denote this matrix by Λ :

$$\Lambda = V^T R^T P R V = \text{diag}(1 - \sigma_1^2, \dots, 1 - \sigma_p^2, 1, \dots, 1).$$

Applying a congruence transformation to the KKT matrix gives

$$\begin{bmatrix} V^T R^T & 0 \\ 0 & U^T \end{bmatrix} \begin{bmatrix} P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} R V & 0 \\ 0 & U \end{bmatrix} = \begin{bmatrix} \Lambda & S^T \\ S & 0 \end{bmatrix},$$

and the inertia of the KKT matrix is equal to the inertia of the matrix on the right.

Applying a permutation to the matrix on the right gives a block diagonal matrix with n diagonal blocks

$$\begin{bmatrix} \lambda_i & \sigma_i \\ \sigma_i & 0 \end{bmatrix}, \quad i = 1, \dots, p, \quad \lambda_i = 1, \quad i = p + 1, \dots, n.$$

The eigenvalues of the 2×2 -blocks are

$$\frac{\lambda_i \pm \sqrt{\lambda_i^2 + 4\sigma_i^2}}{2},$$

i.e., one eigenvalue is positive and one is negative. We conclude that there are $p + (n - p) = n$ positive eigenvalues and p negative eigenvalues.

11.13 *Self-concordance and negative entropy.*

- (a) Show that the negative entropy function $x \log x$ (on \mathbf{R}_{++}) is *not* self-concordant.
 (b) Show that for any $t > 0$, $tx \log x - \log x$ is self-concordant (on \mathbf{R}_{++}).

Solution.

- (a) First we consider $f(x) = x \log x$, for which

$$f'(x) = 1 + \log x, \quad f''(x) = \frac{1}{x}, \quad f'''(x) = -\frac{1}{x^2}.$$

Thus

$$\frac{|f'''(x)|}{f''(x)^{3/2}} = \frac{1/x^2}{1/x^{3/2}} = \frac{1}{\sqrt{x}}$$

which is unbounded above (as $x \rightarrow 0^+$). In particular, the self-concordance inequality $|f'''(x)| \leq 2f''(x)^{3/2}$ fails for $x = 1/5$, so f is *not* self-concordant.

- (b) Now we consider $g(x) = tx \log x - \log x$, for which

$$g'(x) = -\frac{1}{x} + t + t \log x, \quad g''(x) = \frac{1}{x^2} + \frac{t}{x}, \quad g'''(x) = -\frac{2}{x^3} - \frac{t}{x^2}.$$

Therefore

$$\frac{|g'''(x)|}{g''(x)^{3/2}} = \frac{2/x^3 + t/x^2}{(1/x^2 + t/x)^{3/2}} = \frac{2 + tx}{(1 + tx)^{3/2}}.$$

Define

$$h(a) = \frac{2 + a}{(1 + a)^{3/2}}$$

so that

$$h(tx) = \frac{|g'''(x)|}{g''(x)^{3/2}}.$$

We have $h(0) = 2$ and we will show that $h'(a) < 0$ for $a > 0$, *i.e.*, h is decreasing for $a > 0$. This will prove that $h(a) \leq h(0) = 2$, and therefore

$$\frac{|g'''(x)|}{g''(x)^{3/2}} \leq 2.$$

We have

$$\begin{aligned} h'(a) &= \frac{(1 + a)^{3/2} - (3/2)(1 + a)^{1/2}(2 + a)}{(1 + a)^3} \\ &= \frac{(1 + a)^{1/2}((1 + a) - (3/2)(2 + a))}{(1 + a)^3} \\ &= -\frac{(2 + a/2)}{(1 + a)^{5/2}} \\ &< 0, \end{aligned}$$

for $a > 0$, so we are done.

Solutions to additional exercises

Suggestions for exercise 9.30

We recommend the following to generate a problem instance:

```
n = 100;
m = 200;
randn('state',1);
A=randn(m,n);
```

Of course, you should try out your code with different dimensions, and different data as well.

In all cases, be sure that your line search *first* finds a step length for which the tentative point is in $\text{dom } f$; if you attempt to evaluate f outside its domain, you'll get complex numbers, and you'll never recover.

To find expressions for $\nabla f(x)$ and $\nabla^2 f(x)$, use the chain rule (see Appendix A.4); if you attempt to compute $\partial^2 f(x)/\partial x_i \partial x_j$, you will be sorry.

To compute the Newton step, you can use `vnt=-H\g`.

Suggestions for exercise 9.31

For 9.31a, you should try out $N = 1$, $N = 15$, and $N = 30$. You might as well compute and store the Cholesky factorization of the Hessian, and then back solve to get the search directions, even though you won't really see any speedup in Matlab for such a small problem. After you evaluate the Hessian, you can find the Cholesky factorization as `L=chol(H,'lower')`. You can then compute a search step as `-L'\(L\g)`, where \mathbf{g} is the gradient at the current point. Matlab will do the right thing, *i.e.*, it will first solve `L\g` using forward substitution, and then it will solve `-L'\(L\g)` using backward substitution. Each substitution is order n^2 .

To fairly compare the convergence of the three methods (*i.e.*, $N = 1$, $N = 15$, $N = 30$), the horizontal axis should show the approximate total number of flops required, and not the number of iterations. You can compute the approximate number of flops using $n^3/3$ for each factorization, and $2n^2$ for each solve (where each 'solve' involves a forward substitution step and a backward substitution step).

Additional exercises

1. *A structural optimization problem.* Figure 1 shows a two-bar truss with height $2h$ and width w . The two bars are cylindrical tubes with inner radius r and outer radius R . We are interested in determining the values of r , R , w , and h that minimize the weight of the truss subject to a number of constraints. The structure should be strong enough for two loading scenarios. In the first scenario a vertical force F_1 is applied to the node; in the second scenario the force is horizontal with magnitude F_2 .

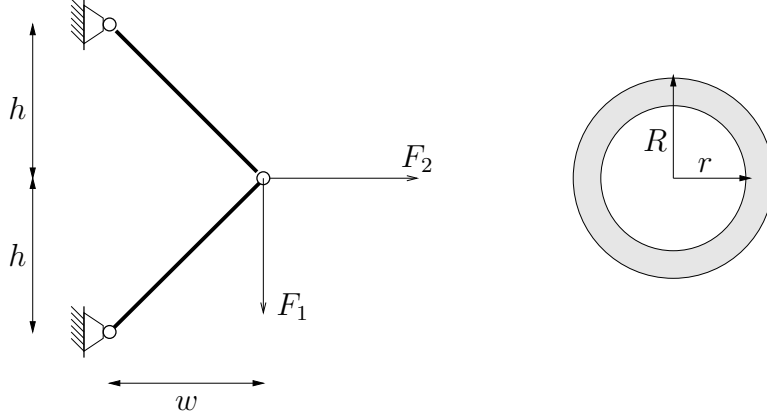


Figure 1 Two-bar truss and cross section of one of the two bars.

The weight of the truss is proportional to the total volume of the bars, which is given by

$$2\pi(R^2 - r^2)\sqrt{w^2 + h^2}$$

This is the cost function in the design problem.

The first constraint is that the truss should be strong enough to carry the load F_1 , *i.e.*, the stress caused by the external force F_1 must not exceed a given maximum value. To formulate this constraint, we first determine the forces in each bar when the structure is subjected to the vertical load F_1 . From the force equilibrium and the geometry of the problem we can determine that the magnitudes of the forces in two bars are equal and given by

$$\frac{\sqrt{w^2 + h^2}}{2h} F_1.$$

The maximum force in each bar is equal to the cross-sectional area times the maximum allowable stress σ (which is a given constant). This gives us the first constraint:

$$\frac{\sqrt{w^2 + h^2}}{2h} F_1 \leq \sigma\pi(R^2 - r^2).$$

The second constraint is that the truss should be strong enough to carry the load F_2 . When F_2 is applied, the magnitudes of the forces in two bars are again equal and given by

$$\frac{\sqrt{w^2 + h^2}}{2w} F_2,$$

which gives us the second constraint:

$$\frac{\sqrt{w^2 + h^2}}{2w} F_2 \leq \sigma\pi(R^2 - r^2).$$

We also impose limits $w_{\min} \leq w \leq w_{\max}$ and $h_{\min} \leq h \leq h_{\max}$ on the width and the height of the structure, and limits $1.1r \leq R \leq R_{\max}$ on the outer radius.

In summary, we obtain the following problem:

$$\begin{aligned}
& \text{minimize} && 2\pi(R^2 - r^2)\sqrt{w^2 + h^2} \\
& \text{subject to} && \frac{\sqrt{w^2 + h^2}}{2h}F_1 \leq \sigma\pi(R^2 - r^2) \\
& && \frac{\sqrt{w^2 + h^2}}{2w}F_2 \leq \sigma\pi(R^2 - r^2) \\
& && w_{\min} \leq w \leq w_{\max} \\
& && h_{\min} \leq h \leq h_{\max} \\
& && 1.1r \leq R \leq R_{\max} \\
& && R > 0, r > 0, w > 0, h > 0.
\end{aligned}$$

The variables are R, r, w, h .

Formulate this as a geometric programming problem.

Solution.

We can introduce new variables

$$u = R^2 - r^2, \quad L = \sqrt{w^2 + h^2}$$

and write the problem as

$$\begin{aligned}
& \text{minimize} && 2\pi uL \\
& \text{subject to} && (F_1/(2\sigma\pi))Lh^{-1}u^{-1} \leq 1 \\
& && (F_2/(2\sigma\pi))Lw^{-1}u^{-1} \leq 1 \\
& && (1/w_{\max})w \leq 1, \quad w_{\min}w^{-1} \leq 1 \\
& && (1/h_{\max})h \leq 1, \quad h_{\min}h^{-1} \leq 1 \\
& && 0.21r^2u^{-1} \leq 1 \\
& && (1/R_{\max}^2)u + (1/R_{\max}^2)r^2 \leq 1 \\
& && w^2L^{-2} + h^2L^{-2} \leq 1 \\
& && w > 0, h > 0, L > 0, r > 0, u > 0,
\end{aligned}$$

with scalar variables h, w, r, u , and L . The desired values can be recovered from the GP by calculating $R = \sqrt{u + r^2}$.

A geometric programming problem can only have *monomial* equality constraints, so we cannot add an equality constraint $L^2 = w^2 + h^2$. Therefore we changed it to an inequality

$$L^2 \geq w^2 + h^2,$$

i.e., $w^2L^{-2} + h^2L^{-2} \leq 1$. To see why this works, notice that L appears only in the objective and the first two inequality constraints. Each of these involves an expression that is monotonically increasing in L , so at the optimum L will be equal to $\sqrt{w^2 + h^2}$. If this is not the case, then we could make L smaller, which maintains feasibility and strictly decreases the objective.

Also, note that we replaced the inequality $R \geq 1.1r$ with $u \geq (1.1^2 - 1)r^2 = 0.21r^2$.

2. *Bounding object position from multiple camera views.* A small object is located at unknown position $x \in \mathbf{R}^3$, and viewed by a set of m cameras. Our goal is to find a box in \mathbf{R}^3 ,

$$\mathcal{B} = \{z \in \mathbf{R}^3 \mid l \preceq z \preceq u\},$$

for which we can guarantee $x \in \mathcal{B}$. We want the smallest possible such bounding box. (Although it doesn't matter, we can use volume to judge 'smallest' among boxes.)

Now we describe the cameras. The object at location $x \in \mathbf{R}^3$ creates an image on the image plane of camera i at location

$$v_i = \frac{1}{c_i^T x + d_i} (A_i x + b_i) \in \mathbf{R}^2.$$

The matrices $A_i \in \mathbf{R}^{2 \times 3}$, vectors $b_i \in \mathbf{R}^2$ and $c_i \in \mathbf{R}^3$, and real numbers $d_i \in \mathbf{R}$ are known, and depend on the camera positions and orientations. We assume that $c_i^T x + d_i > 0$. The 3×4 matrix

$$P_i = \begin{bmatrix} A_i & b_i \\ c_i^T & d_i \end{bmatrix}$$

is called the *camera matrix* (for camera i). It is often (but not always) the case that the first 3 columns of P_i (*i.e.*, A_i stacked above c_i^T) form an orthogonal matrix, in which case the camera is called *orthographic*.

We do not have direct access to the image point v_i ; we only know the (square) pixel that it lies in. In other words, the camera gives us a measurement \hat{v}_i (the center of the pixel that the image point lies in); we are guaranteed that

$$\|v_i - \hat{v}_i\|_\infty \leq \rho_i/2,$$

where ρ_i is the pixel width (and height) of camera i . (We know nothing else about v_i ; it could be any point in this pixel.)

Given the data $A_i, b_i, c_i, d_i, \hat{v}_i, \rho_i$, we are to find the smallest box \mathcal{B} (*i.e.*, find the vectors l and u) that is guaranteed to contain x . In other words, find the smallest box in \mathbf{R}^3 that contains all points consistent with the observations from the camera.

- (a) Explain how to solve this using convex or quasiconvex optimization. You must explain any transformations you use, any new variables you introduce, etc. If the convexity or quasiconvexity of any function in your formulation isn't obvious, be sure to justify it.

- (b) Solve the specific problem instance given in the file `camera_data.m`. Be sure that your final numerical answer (*i.e.*, l and u) stands out.

Solution.

- (a) We get a subset $\mathcal{P} \subseteq \mathbf{R}^3$ (which we'll soon see is a polyhedron) of locations x that are consistent with the camera measurements. To find the smallest box that covers *any* subset in \mathbf{R}^3 , all we need to do is maximize and minimize the (linear) functions x_1 , x_2 , and x_3 to get l and u . Here \mathcal{P} is a polyhedron, so we'll end up solving 6 LPs, one to get each of l_1 , l_2 , l_3 , u_1 , u_2 , and u_3 .

Now let's look more closely at \mathcal{P} . Our measurements tell us that

$$\hat{v}_i - (\rho_i/2)\mathbf{1} \preceq \frac{1}{c_i^T x + d_i} (A_i x + b_i) \preceq \hat{v}_i + (\rho_i/2)\mathbf{1}, \quad i = 1, \dots, m.$$

We multiply through by $c_i^T x + d_i$, which is positive, to get

$$(\hat{v}_i - (\rho_i/2)\mathbf{1})(c_i^T x + d_i) \preceq A_i x + b_i \preceq (\hat{v}_i + (\rho_i/2)\mathbf{1})(c_i^T x + d_i), \quad i = 1, \dots, m,$$

a set of $2m$ linear inequalities on x . In particular, it defines a polyhedron.

To get l_k we solve the LP

$$\begin{aligned} & \text{minimize} && x_k \\ & \text{subject to} && (\hat{v}_i - (\rho_i/2)\mathbf{1})(c_i^T x + d_i) \preceq A_i x + b_i, \quad i = 1, \dots, m, \\ & && A_i x + b_i \preceq (\hat{v}_i + (\rho_i/2)\mathbf{1})(c_i^T x + d_i), \quad i = 1, \dots, m, \end{aligned}$$

for $k = 1, 2, 3$; to get u_k we maximize the same objective.

- (b) Here is a script that solves given instance:

```
% load the data
camera_data;
A1 = P1(1:2,1:3); b1 = P1(1:2,4); c1 = P1(3,1:3); d1 = P1(3,4);
A2 = P2(1:2,1:3); b2 = P2(1:2,4); c2 = P2(3,1:3); d2 = P2(3,4);
A3 = P3(1:2,1:3); b3 = P3(1:2,4); c3 = P3(3,1:3); d3 = P3(3,4);
A4 = P4(1:2,1:3); b4 = P4(1:2,4); c4 = P4(3,1:3); d4 = P4(3,4);

cvx_quiet(true);
for bounds = 1:6
    cvx_begin
        variable x(3)
        switch bounds
            case 1
                minimize x(1)
            case 2
                maximize x(1)
```

```

        case 3
            minimize x(2)
        case 4
            maximize x(2)
        case 5
            minimize x(3)
        case 6
            maximize x(3)
    end
    % constraints for 1st camera
    (vhat(:,1)-rho(1)/2)*(c1*x + d1) <= A1*x + b1;
    A1*x + b1 <= (vhat(:,1)+rho(1)/2)*(c1*x + d1);
    % constraints for 2ns camera
    (vhat(:,2)-rho(2)/2)*(c2*x + d2) <= A2*x + b2;
    A2*x + b2 <= (vhat(:,2)+rho(2)/2)*(c2*x + d2);
    % constraints for 3rd camera
    (vhat(:,3)-rho(3)/2)*(c3*x + d3) <= A3*x + b3;
    A3*x + b3 <= (vhat(:,3)+rho(3)/2)*(c3*x + d3);
    % constraints for 4th camera
    (vhat(:,4)-rho(4)/2)*(c4*x + d4) <= A4*x + b4;
    A4*x + b4 <= (vhat(:,4)+rho(4)/2)*(c4*x + d4);
    cvx_end
    val(bounds) = cvx_optval;
end
disp(['l1 = ' num2str(val(1))]);
disp(['l2 = ' num2str(val(3))]);
disp(['l3 = ' num2str(val(5))]);
disp(['u1 = ' num2str(val(2))]);
disp(['u2 = ' num2str(val(4))]);
disp(['u3 = ' num2str(val(6))]);

```

The script returns the following results:

```

l1 = -0.99561
l2 = 0.27531
l3 = -0.67899
u1 = -0.8245
u2 = 0.37837
u3 = -0.57352

```

3. *Bounding portfolio risk with incomplete covariance information.* Consider the following instance of the problem described in §4.6, on p171–173 of *Convex Optimization*. We suppose that Σ_{ii} , which are the squares of the price volatilities of the assets, are known.

For the off-diagonal entries of Σ , all we know is the sign (or, in some cases, nothing at all). For example, we might be given that $\Sigma_{12} \geq 0$, $\Sigma_{23} \leq 0$, etc. This means that we do not know the correlation between p_1 and p_2 , but we do know that they are nonnegatively correlated (*i.e.*, the prices of assets 1 and 2 tend to rise or fall together).

Compute σ_{wc} , the worst-case variance of the portfolio return, for the specific case

$$x = \begin{bmatrix} 0.1 \\ 0.2 \\ -0.05 \\ 0.1 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 0.2 & + & + & \pm \\ + & 0.1 & - & - \\ + & - & 0.3 & + \\ \pm & - & + & 0.1 \end{bmatrix},$$

where a “+” entry means that the element is nonnegative, a “−” means the entry is nonpositive, and “±” means we don’t know anything about the entry. (The negative value in x represents a *short position*: you sold stocks that you didn’t have, but must produce at the end of the investment period.) In addition to σ_{wc} , give the covariance matrix Σ_{wc} associated with the maximum risk. Compare the worst-case risk with the risk obtained when Σ is diagonal.

Hint. To handle positive semidefiniteness of a matrix $X \in \mathbf{S}^n$ in CVX, use `X == semidefinite(n)`; see the user guide.

Solution. We can solve the problem with the following CVX code:

```
x = [0.1 0.2 -0.05 0.1]';

cvx_begin
    variable Sigma(4,4) symmetric
    maximize(x'*Sigma*x)
    subject to
        Sigma(1,1) == 0.2; Sigma(2,2) == 0.1;
        Sigma(3,3) == 0.3; Sigma(4,4) == 0.1;
        Sigma(1,2) >= 0; Sigma(1,3) >= 0;
        Sigma(2,3) <= 0; Sigma(2,4) <= 0;
        Sigma(3,4) >= 0
        Sigma == semidefinite(4)
cvx_end

s_wc = sqrt(cvx_optval)
```

Running this script we get the optimal value to be 0.0151662, that is, we get $\sigma_{\text{wc}} = 0.1232$. The associated Σ matrix is given by

$$\Sigma = \begin{bmatrix} 0.2000 & 0.0790 & 0.0000 & 0.1118 \\ 0.0790 & 0.1000 & -0.1387 & 0.0000 \\ 0.0000 & -0.1387 & 0.3000 & 0.0752 \\ 0.1118 & 0.0000 & 0.0752 & 0.1000 \end{bmatrix}.$$

4. *Three-way linear classification.* We are given data

$$x^{(1)}, \dots, x^{(N)}, \quad y^{(1)}, \dots, y^{(M)}, \quad z^{(1)}, \dots, z^{(P)},$$

three nonempty sets of vectors in \mathbf{R}^n . We wish to find three affine functions on \mathbf{R}^n ,

$$f_i(z) = a_i^T z - b_i, \quad i = 1, 2, 3,$$

that satisfy the following properties:

$$\begin{aligned} f_1(x^{(j)}) &> \max\{f_2(x^{(j)}), f_3(x^{(j)})\}, & j = 1, \dots, N, \\ f_2(y^{(j)}) &> \max\{f_1(y^{(j)}), f_3(y^{(j)})\}, & j = 1, \dots, M, \\ f_3(z^{(j)}) &> \max\{f_1(z^{(j)}), f_2(z^{(j)})\}, & j = 1, \dots, P. \end{aligned}$$

In words: f_1 is the largest of the three functions on the x data points, f_2 is the largest of the three functions on the y data points, f_3 is the largest of the three functions on the z data points. We can give a simple geometric interpretation: The functions f_1 , f_2 , and f_3 partition \mathbf{R}^n into three regions,

$$\begin{aligned} R_1 &= \{z \mid f_1(z) > \max\{f_2(z), f_3(z)\}\}, \\ R_2 &= \{z \mid f_2(z) > \max\{f_1(z), f_3(z)\}\}, \\ R_3 &= \{z \mid f_3(z) > \max\{f_1(z), f_2(z)\}\}, \end{aligned}$$

defined by where each function is the largest of the three. Our goal is to find functions with $x^{(j)} \in R_1$, $y^{(j)} \in R_2$, and $z^{(j)} \in R_3$.

Pose this as a convex optimization problem. You may not use strict inequalities in your formulation.

Solve the specific instance of the 3-way separation problem given in `sep3way_data.m`, with the columns of the matrices \mathbf{X} , \mathbf{Y} and \mathbf{Z} giving the $x^{(j)}$, $j = 1, \dots, N$, $y^{(j)}$, $j = 1, \dots, M$ and $z^{(j)}$, $j = 1, \dots, P$. To save you the trouble of plotting data points and separation boundaries, we have included the plotting code in `sep3way_data.m`. (Note that `a1`, `a2`, `a3`, `b1` and `b2` contain arbitrary numbers; you should compute the correct values using `cvx`.)

Solution. The inequalities

$$\begin{aligned} f_1(x^{(j)}) &> \max\{f_2(x^{(j)}), f_3(x^{(j)})\}, & j = 1, \dots, N, \\ f_2(y^{(j)}) &> \max\{f_1(y^{(j)}), f_3(y^{(j)})\}, & j = 1, \dots, M, \\ f_3(z^{(j)}) &> \max\{f_1(z^{(j)}), f_2(z^{(j)})\}, & j = 1, \dots, P. \end{aligned}$$

are homogeneous in a_i and b_i so we can express them as

$$\begin{aligned} f_1(x^{(j)}) &\geq \max\{f_2(x^{(j)}), f_3(x^{(j)})\} + 1, & j = 1, \dots, N, \\ f_2(y^{(j)}) &\geq \max\{f_1(y^{(j)}), f_3(y^{(j)})\} + 1, & j = 1, \dots, M, \\ f_3(z^{(j)}) &\geq \max\{f_1(z^{(j)}), f_2(z^{(j)})\} + 1, & j = 1, \dots, P. \end{aligned}$$

Note that we can add any vector α to each of the a_i , without affecting these inequalities (which only refer to difference between a_i 's), and we can add any number β to each of the b_i 's for the same reason. We can use this observation to normalize or simplify the a_i and b_i . For example, we can assume without loss of generality that $a_1 + a_2 + a_3 = 0$ and $b_1 + b_2 + b_3 = 0$.

The following script implements this method for 3-way classification and tests it on a small separable data set

```
clear all; close all;
sep3way_data;

cvx_begin
variables a1(2) a2(2) a3(2) b1 b2 b3
    a1'*X-b1 >= max(a2'*X-b2,a3'*X-b3)+1;
    a2'*Y-b2 >= max(a1'*Y-b1,a3'*Y-b3)+1;
    a3'*Z-b3 >= max(a1'*Z-b1,a2'*Z-b2)+1;
    a1 + a2 + a3 == 0
    b1 + b2 + b3 == 0
cvx_end

% now plot the points, and the three-way separation induced
% by a1, a2, a3, b1, b2, b3
% find maximally confusing point
p = [(a1-a2)';(a1-a3)']\[(b1-b2);(b1-b3)];

% plot
t = [-7:0.01:7];
u1 = a1-a2; u2 = a2-a3; u3 = a3-a1;
v1 = b1-b2; v2 = b2-b3; v3 = b3-b1;
line1 = (-t*u1(1)+v1)/u1(2); idx1 = find(u2'*[t;line1]-v2>0);
line2 = (-t*u2(1)+v2)/u2(2); idx2 = find(u3'*[t;line2]-v3>0);
line3 = (-t*u3(1)+v3)/u3(2); idx3 = find(u1'*[t;line3]-v1>0);
plot(X(1,:),X(2,:), 'r', Y(1,:), Y(2,:), 'ro', Z(1,:), Z(2,:), 'g+', ...
    t(idx1), line1(idx1), 'k', t(idx2), line2(idx2), 'k', t(idx3), line3(idx3), 'k');
axis([-7 7 -7 7]);
```

The following figure is generated.

