CONVEX OPTIMIZATION HOMEWORK 6

TOMMENIX YU
ID: 12370130
STAT 31015
DUE WED FEB 22, 2023, 3PM

Exercise 1.

Proof.

(a)$\underline{X \succ 0 \iff A \succ 0, S \succ 0}$:

Note that row operations does not affect the eigenvalues of a matrix, thus we have

$$X = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix} = \begin{pmatrix} A & B \\ B^T - B^T A^{-1} A & C - B^T A^{-1} B \end{pmatrix} = \begin{pmatrix} A & B \\ 0 & S \end{pmatrix}$$

and so the eigenvalues of X are those of A and of S. This means that if all eigenvalues of X are positive, it is equivalent to all eigenvalues of A and S are positive. Thus the result follows.

(b)Show the Woodbury formula:

We have

$$\begin{pmatrix} A & B \\ B^T & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ b \end{pmatrix}$$

where we assume C is invertible to use Gaussian elimination to cancel the right top block we get

$$\begin{pmatrix} A - BC^{-1}B^T & 0 \\ B^T & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -BC^{-1}b \\ b \end{pmatrix}$$

where the first block row gives us

$$(A - BC^{-1}B^T)x = -BC^{-1}b \ \Rightarrow B^T x = -B^T(A - BC^{-1}B^T)^{-1}BC^{-1}b$$

and the second row gives us

$$B^T x + Cy = b \ \Rightarrow B^T x = b - Cy$$

combining both we get

$$b - Cy = -B^T(A - BC^{-1}B^T)^{-1}BC^{-1}b$$
$$\Rightarrow y = (C^{-1} - C^{-1}B^T(A - BC^{-1}B^T)^{-1}BC^{-1})b$$

and thus

$$W = C^{-1} - C^{-1}B^T(A - BC^{-1}B^T)^{-1}BC^{-1}.$$

(c)Operation count:

We know A is $n \times n$, B is $n \times m$, C is $m \times m$.

To compute directly, we will need to form S, then compute the inverse.

To form S, we have

- $\operatorname{flop}(A^{-1} \cdot B) = mn(2n-1) = O(2mn^2)$;
- $\operatorname{flop}(B^T \cdot (A^{-1}B)) = m^2(2n-1) = O(2m^2n)$;
- $\operatorname{flop}(C - (B^T A^{-1} B)) = m^2 = O(m^2)$;

and so in total $O(4m^2n)$ flops. And finding the inverse of $m \times m$ matrices costs $O(m^3)$ flops, so the total flop is $O(m^3) + O(4m^2n) = O(m^3)$ for $m \gg n$.

For the Woodbury formula, we know that it includes forming the matrix $A - BC^{-1}B^T$, getting its inverse, getting the inverse of C, and getting all the matrix multiplications.

- $\operatorname{flop} C^{-1} = O(m)$;
- $\operatorname{flop} A - BC^{-1}B^T = n^2 + 2mn^2 - n^2 + 2m^2n - mn = O(2m^2n)$;
- $\operatorname{flop}(A - BC^{-1}B^T)^{-1} = O(n^3)$;
- Since we've formed $C^{-1}B$ already above

$$\begin{aligned}\operatorname{flop} W &= \operatorname{flop}(-_{m\times m}) + \operatorname{flop}((C^{-1}B^T)(A - BC^{-1}B^T)^{-1}BC^{-1}) \\ &= m^2 + O(6m^2n) = O(6m^2n)\end{aligned}$$

and thus this method needs in total $O(8m^2n)$ flops, which is significantly smaller than $O(m^3)$ since $m \gg n$.

$\square$

Exercise 2.

Proof.

Denote

$$y_i = \exp(a_i^T x + b_i)$$

then we have

$$\nabla f = x + \frac{\sum_{i=1}^m y_i a_i}{\sum_{i=1}^m y_i} = x + \frac{\sum_{i=1}^m y_i}{\sum_{i=1}^m y_i} a_i$$

and

$$\nabla^2 f = I + \frac{\left(\sum_{i=1}^m y_i a_i a_i^T\right)\left(\sum_{i=1}^m y_i\right) - \left(\sum_{i=1}^m y_i a_i\right)\left(\sum_{j=1}^m y_j a_j\right)^T}{\left(\sum_{i=1}^m y_i\right)^2}.$$

The Newton equation is

$$-\nabla^2 f(x)\Delta x = \nabla f(x_n)$$

to solve which we need to find a way to invert the Hessian of f. So we try to write it in a form like that of prob 1 to get (since $A^T = \left(a_1, \dots, a_n\right))$

$$\nabla^2 f = I + \frac{1}{\left(\sum_{i=1}^m y_i\right)^2}\left(A^T\left(\left(\sum_{i=1}^m y_i\right) \cdot Y_1\right) A + A^T Y_2 A\right) = I - A^T B A$$

where

$$Y_1 = (y_{i,j}^1) : y_{i,j}^1 = \begin{cases} 0 & i \neq j \\ y_i & i = j \end{cases}, \quad and \quad Y_2 = yy^T$$

and

$$B = -\frac{1}{\left(\sum_{i=1}^m y_i\right)^2}\left[\left(\sum_{i=1}^m y_i\right) \cdot Y_1 + Y_2\right].$$

Forming each matrix inside cost at most $O(m^3)$ where as computing the inverse cost $O(n^2 m)$. So the total leading order is then $O(n^2 m)$.

□

Exercise 3.

Proof.

The sequence is monotone:

Use x to denote x_n, and h denotes Δx_n.

Using Taylor on the derivative we have

$$0 = f'(x*) = f'(x) + (x^* - x)f''(x) + \frac{(x^* - x)^2}{2} f'''(s) \leq f'(x) + (x^* - x)f''(x)$$
$$\Rightarrow -f'(x) \leq (x^* - x)f''(x)$$

but then we have (plugging in definition of h)

$$x + h = x - \frac{f'(x)}{f''(x)} \leq x + \frac{(x^* - x)f''(x)}{f''(x)} = x^*.$$

This means that we're always on the left part of x^*. Moreover, by definition of the Newton step we are guaranteed to go down, which means that we move toward right each time, thus x_n is monotone.

The line search can always end at $t = 1$:

What the statement means is that for all $\alpha \in (0, 1/2)$ we always have (use x to denote x_n, and h denotes Δx_n)

$$f(x + h) < f(x) + \alpha \nabla f^T(x)h.$$

By Taylor's residue formula we have

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) \leq f(x) + hf'(x) + \frac{h^2}{2} f''(x)$$

since $f'''(t) \leq 0$. But the last 2 terms by construction is

$$hf'(x) + \frac{h^2}{2} f''(x) = \frac{1}{2} \nabla f^T(x)h$$

where since $\nabla f^T(x)h = f'(x)h < 0$ we have that the inf of the right hand side over α is attained at $\alpha \to \frac{1}{2}$, which then means that

$$f(x + h) < f(x) + \alpha \nabla f^T(x)h$$

for all α.

$\square$

Exercise 4.

Proof.

(1)Rephrase, gradient and Hessian:

The question is just to minimize the negative of the expression, i.e.

$$\min \sum_{i=1}^{m} \log\left(1+\exp(a^T u_i + b)\right) - \sum_{i=1}^{q}(a^T u_i + b)$$

for which we compute the gradient and Hessian.

$$\frac{\partial f}{\partial a} = \sum^{m} \frac{u_i e^{a^T u_i + b}}{1+e^{a^T u_i + b}} - \sum^{q} u_i$$

and

$$\frac{\partial f}{\partial b} = \sum^{m} \frac{e^{a^T u_i + b}}{1+e^{a^T u_i + b}} - q.$$

For the second derivative we have

$$\frac{\partial^2 f}{\partial a^2} = \sum^{m} \frac{u_i u_i^T e^{a^T u_i + b}}{\left(1+e^{a^T u_i + b}\right)^2}; \quad \frac{\partial^2 f}{\partial a \partial b} = \sum^{m} \frac{u_i e^{a^T u_i + b}}{\left(1+e^{a^T u_i + b}\right)^2}; \quad \frac{\partial^2 f}{\partial b^2} = \sum^{m} \frac{e^{a^T u_i + b}}{\left(1+e^{a^T u_i + b}\right)^2}.$$

And thus

$$\nabla f = \begin{pmatrix} \sum^m \frac{u_i e^{a^T u_i + b}}{1+e^{a^T u_i + b}} - \sum^q u_i \\ \sum^m \frac{e^{a^T u_i + b}}{1+e^{a^T u_i + b}} - q \end{pmatrix}$$

and

$$\nabla^2 f = \begin{pmatrix} \sum^m \frac{u_i u_i^T e^{a^T u_i + b}}{\left(1+e^{a^T u_i + b}\right)^2} & \sum^m \frac{u_i e^{a^T u_i + b}}{\left(1+e^{a^T u_i + b}\right)^2} \\ \sum^m \frac{u_i^T e^{a^T u_i + b}}{\left(1+e^{a^T u_i + b}\right)^2} & \sum^m \frac{e^{a^T u_i + b}}{\left(1+e^{a^T u_i + b}\right)^2} \end{pmatrix}$$

where if we let

$$v_i := \frac{e^{(a^T u_i + b)/2}}{1+e^{a^T u_i + b}} \begin{pmatrix} u_i \\ 1 \end{pmatrix}$$

then we have

$$\nabla^2 f = \sum_{i=1}^{m} v_i v_i^T.$$

(2): The code for function implementation:

```
function [val] = tomin(X,q,a,b)
% Just the value of the function to min; Here, using the convention from
% the code given assume X to be the mx(n+1) matrix with the last column
% being 1s. q is the number of true indices, and a,b are initial guesses.

[m,nn] = size(X);
n = nn-1;
%specifies the dimension for later use.

val = 0;
for i = 1: m
    ee = exp(a.'*X(i,1:n)+b);
    val = val + log(1+ee);
end

for i = 1:q
    val = val - (a.'*X(i,1:n)+b);
end
end
```

and for gradient and Hessian

```
function [grad,hess] = GradHess(X,q,a,b)
% Just the Gradient and Hessian of the function to min; Here, using the
% convention from
% the code given assume X to be the mx(n+1) matrix with the last column
% being 1s. q is the number of true indices, and a,b are initial guesses.


[m,nn] = size(X);
n = nn-1;
%specifies the dimension for later use.

gradup = zeros(nn-1,1);
%upper part of gradient
gradlower = zeros(1,1);

ee = zeros(m,1);
eee = zeros(m,1);
for i = 1:m
    s = a.'*X(i,1:n)+b;
    ee(i) = exp(s)/(1+exp(s));
    eee(i) = exp(s/2)/(1+exp(s));
end

for i = 1:m
    gradup = gradup + ee(i)*(X(i,1:n).');
    gradlower = gradlower+ee(i);
end

for i = 1:q
    gradup = gradup - X(i,1:n).';
    gradlower = gradlower -1;
end

grad = [gradup; gradlower];

%Now we do Hessian using the method of vv^T
hess = zeros(nn,nn);
for i = 1:m
    v = eee(i)*(X(i,:).');
    hess = hess + v*v.';
end
end
```

(c): The result is (for the fixed random state)

$m = 100$

```
>> test

aa100 =

    0.9848


bb100 =

   -4.9820
```

$m = 200$

```
>> test

aa200 =

    0.9848


bb200 =

   -4.9820

>>
```

$m = 400$

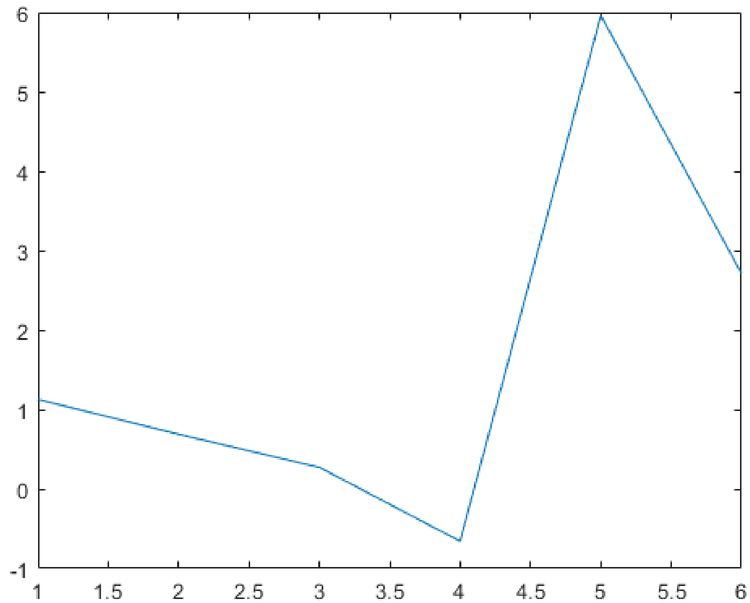
```
>> test

aa400 =

    0.9848


bb400 =

   -4.9820
```

and the code is below (note that to change m we need to change the m specified in the file, since the form we're asked to do takes only 2 arguments, a and b. But we can of course move m as an argument too...):

```
function [a,b] = hwk6p4(a0,b0)
% This function gives the finalized result of Newton's method, with initial
% guess a0, b0. Here, epsilon is 0.000000001 and the setting is from data.

alpha = 0.49;
beta = 0.9;
epsilon = 0.00000001;

m = 400;
u = 10*rand(m,1);
y = (rand(m,1) < exp(a0*u+b0)./(1+exp(a0*u+b0)));

% order the observation data
ind_false = find( y == 0 );
ind_true  = find( y == 1 );

% X is the sorted design matrix
% first have true than false observations followed by the bias term
X = [u(ind_true); u(ind_false)];
X = [X ones(size(u,1),1)];
[m,n] = size(X);
q = length(ind_true);

[gr,he] = GradHess(X,q,a0,b0);

lambda2 = gr.'*inv(he)*gr;
a = a0;
b = b0;

while lambda2 > 2*epsilon
    step = -inv(he)*gr;
    t=1;
    a1 = a + t*step(1);
    b1 = b + t*step(2);
    while tomin(X,q,a1,b1) >= tomin(X,q,a,b) +alpha*t*gr.'*step
        t = t*beta;
        a1 = a + t*step(1);
        b1 = b + t*step(2);
    end
    a = a + t*step(1);
    b = b + t*step(2);
    [gr,he] = GradHess(X,q,a,b);
    lambda2 = gr.'*inv(he)*gr;
end
end
```

(4):

The graph for the whole iterate is



which makes sense because in class we know that when the difference is small enough the decay is extremely rapid.

□