STAT 375 Homework 5 Solutions

Problem (1)

One can rewrite the cost function in the following fashion:

$$\begin{aligned}\mathcal{C}_{A,y}(x=(z,s)) &= -\frac{1}{2}\begin{pmatrix} z & s \end{pmatrix}\begin{pmatrix} I & A \\ A^T & -\lambda I \end{pmatrix}\begin{pmatrix} z \\ s \end{pmatrix} + \begin{pmatrix} y & 0 \end{pmatrix}\begin{pmatrix} z \\ s \end{pmatrix} \\ &= \frac{1}{2}\langle x, Qx\rangle + \langle b, x\rangle\end{aligned}$$

where $Q := -\begin{pmatrix} I & A \\ A^T & -\lambda I \end{pmatrix}$ and $b := \begin{pmatrix} y \\ 0 \end{pmatrix}$.

Problem (2)

The stationary point of the quadratic form is given by:

$$\begin{aligned}\nabla\mathcal{C}_{A,y}(x) &= Qx + b \\ \Rightarrow \hat{x} &= -Q^{-1}b\end{aligned}$$

In terms of the original variables we get:

$$\begin{pmatrix} \hat{z} \\ \hat{s} \end{pmatrix} = \begin{pmatrix} I & A \\ A^T & -\lambda I \end{pmatrix}^{-1}\begin{pmatrix} y \\ 0 \end{pmatrix}$$

By block inversion we obtain:

$$\hat{s} = (\lambda I + A^T A)^{-1} A^T y$$

which is the ridge regression estimator. The block inversion is done by eliminating $\hat{z}$ as follows:

$$\begin{aligned}\hat{z} + A\hat{s} &= y \\ \Rightarrow \hat{z} &= y - As\end{aligned}$$

Now using the second equation $A^T\hat{z} - \lambda\hat{s} = 0$, we have $\hat{s} = (\lambda I + A^T A)^{-1} A^T y$.

Problem (3)

The belief propagation update equations are as follows:

$$\begin{aligned}\alpha_{i\to j}^{(t+1)} &= -Q_{ii} - \sum_{k\in\partial i\setminus j} \frac{Q_{ik}^2}{\alpha_{k\to i}^{(t)}} \\ \beta_{i\to j}^{(t+1)} &= -b_i + \sum_{k\in\partial i\setminus j} \frac{Q_{ik}\beta_{k\to i}^{(t)}}{\alpha_{k\to i}^{(t)}}\end{aligned}$$

Problem (4)

With increasing noise, the residuals are larger, thus a better behavior is observed when we use $\lambda = \sigma^2$. This effectively normalizes the weight of the residues. Indeed, the algorithm does not converge when one uses $\lambda = \frac{1}{\sigma^2}$. Fig. 1 shows the convergence of the algorithm within 50 iterations. The mean square error of the ridge estimate computed via BP is plotted in Fig. 2. The code that implements the algorithm is

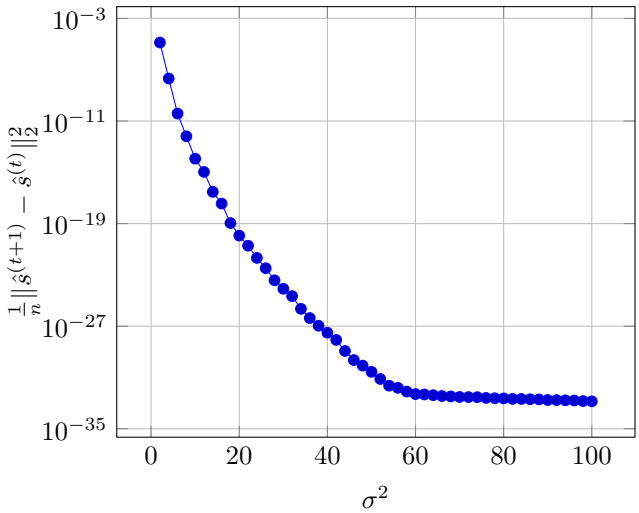

Figure 1: Convergence of the BP algorithm for different values of σ^2

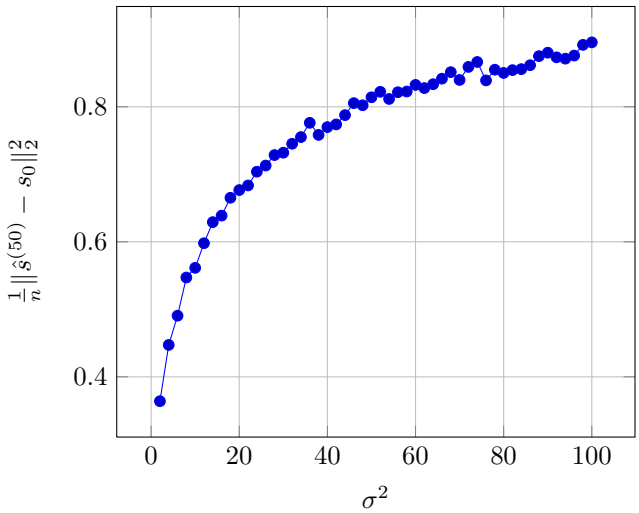

Figure 2: The mean square error of the (approximate) ridge regression estimator obtained via BP

as follows:

```
clear all
m = 800;
n = 1000;

l = 20;
i = kron(1:m, ones(1, l));
j = zeros(1, length(i));
```

```
for iter = 1:m
    y = randsample(n, l)';
    j(((iter-1)*l+1):(iter-1)*l+l) = y;
end

k = 2*double(rand(1, length(i)) >= 0.5) -1;

A = sparse(i, j, k, m, n);
[i, j, k] = find(A);
nz = nnz(A);


% Q = [zeros(m,m), A; A' zeros(n, n)];
% Q = sparse(Q);

% [iq, jq, kq] = find(Q);
% nz = nnz(Q);

uniqid = (m+n)*i + j; %This gives iq,jq uniquely
id = [1:length(uniqid)];

mapobj = containers.Map(uniqid,id);

iedgeadjc = zeros(1000000,1);
jedgeadjc = zeros(1000000,1);
iedgeadjv = zeros(1000000,1);
jedgeadjv = zeros(1000000,1);

entryedgeadjbetac =zeros(1e6,1);
entryedgeadjbetav = zeros(1e6,1);
%Edge adjacency for bp updates
cnt = 0;
cnt1 = 0;
for ctr=1:m
    ctr
    q1 = find(A(ctr,:));
    for ctr1= q1
        edgeid = mapobj([(m+n)*ctr+ctr1]);
        othernodes = setdiff(q1,[ctr1]);
        cnt1=cnt1+1;
        for ctr2 = othernodes
            edgeconn = mapobj([(m+n)*ctr+ctr2]);
            cnt=cnt+1;
            iedgeadjc(cnt)=edgeid;
            jedgeadjc(cnt)=edgeconn;
            entryedgeadjbetac(cnt) = -A(ctr,ctr2);
            %entryedgeadjbeta(cnt) = -Q(ctr2,ctr);
        end
    end
end
edgeadjalphc = sparse(iedgeadjc(1:cnt),jedgeadjc(1:cnt),-1);
edgeadjbetac = sparse(iedgeadjc(1:cnt),jedgeadjc(1:cnt),entryedgeadjbetac(1:cnt));


cnt = 0;
cnt1 = 0;
for ctr=1:n
    ctr
    q1 = find(A(:,ctr));
    for ctr1 = q1'
        %[ctr1,ctr, (m+n)*ctr1+ctr]
        edgeid = mapobj([(m+n)*ctr1+ctr]);
        othernodes = setdiff(q1,[ctr1]);
        cnt1=cnt1+1;
        for ctr2 = othernodes'
            edgeconn = mapobj([(m+n)*ctr2+ctr]);
            cnt=cnt+1;
            iedgeadjv(cnt)=edgeid;
            jedgeadjv(cnt)=edgeconn;
            entryedgeadjbetav(cnt) = -A(ctr2,ctr);
```

```
            %entryedgeadjbeta(cnt) = -Q(ctr2,ctr);
        end
    end
end
edgeadjalphv = sparse(iedgeadjv(1:cnt),jedgeadjv(1:cnt),-1);
edgeadjbetav = sparse(iedgeadjv(1:cnt),jedgeadjv(1:cnt),entryedgeadjbetav(1:cnt));

%Edgeadjacency for final node expressions

iedgeadjcf = zeros(1000000,1);
jedgeadjcf = zeros(1000000,1);
iedgeadjvf = zeros(1000000,1);
jedgeadjvf = zeros(1000000,1);

entryedgeadjbetacf =zeros(1e6,1);
entryedgeadjbetavf = zeros(1e6,1);
%Edge adjacency for bp updates
cnt = 0;
cnt1 = 0;
for ctr=1:m
    ctr
    q1 = find(A(ctr,:));

        for ctr2 = q1
            edgeconn = mapobj([(m+n)*ctr+ctr2]);
            cnt=cnt+1;
            iedgeadjcf(cnt)=ctr;
            jedgeadjcf(cnt)=edgeconn;
            entryedgeadjbetacf(cnt) = -A(ctr,ctr2);
            %entryedgeadjbeta(cnt) = -Q(ctr2,ctr);
        end

end
edgeadjalphcf = sparse(iedgeadjcf(1:cnt),jedgeadjcf(1:cnt),-1);
edgeadjbetacf = sparse(iedgeadjcf(1:cnt),jedgeadjcf(1:cnt),entryedgeadjbetacf(1:cnt));


cnt = 0;
cnt1 = 0;
for ctr=1:n
    ctr
    q1 = find(A(:,ctr));

        for ctr2 = q1'
            edgeconn = mapobj([(m+n)*ctr2+ctr]);
            cnt=cnt+1;
            iedgeadjvf(cnt)=ctr;
            jedgeadjvf(cnt)=edgeconn;
            entryedgeadjbetavf(cnt) = -A(ctr2,ctr);
            %entryedgeadjbeta(cnt) = -Q(ctr2,ctr);
        end
end
edgeadjalphvf = sparse(iedgeadjvf(1:cnt),jedgeadjvf(1:cnt),-1);
edgeadjbetavf = sparse(iedgeadjvf(1:cnt),jedgeadjvf(1:cnt),entryedgeadjbetavf(1:cnt));
save('decoder2.mat');
load('decoder2.mat');

%now generate the samples

sigma2 = 2:2:100;
conv = zeros(length(sigma2), 1);
mse = conv;
mc = 20;
for iter = 1:length(sigma2)
    iter
    for iter2 = 1:mc
    sigma = sqrt(sigma2(iter));
    lambda = sigma2(iter);
    s0 = randn(n, 1);
    w = sigma*randn(m, 1);
```

```
    y = A*s0+w;
    out = y(i);
    %now do the iterations
    alph2c = ones(16000,1);
    beta2c = zeros(16000,1);

    alph2v = ones(16000,1);
    beta2v = zeros(16000,1);



    for ctr=1:50
        alphprev2c = alph2c;
        betaprev2c = beta2c;
        alphprev2v = alph2v;
        betaprev2v = beta2v;

        alph2v = 1+edgeadjalphc*(1./alphprev2c);
        beta2v = out + edgeadjbetac*(betaprev2c./alphprev2c);
        alph2c = -lambda + edgeadjalphv*(1./alphprev2v);
        beta2c = 0 + edgeadjbetav*(betaprev2v./alphprev2v);

    end

    %Compute final estimates
    solridge = (A'*A+lambda*eye(n))\(A'*y);
    % % % norm(solridge-bpestimatemean(m+1:n+m))

    alphv = -lambda + edgeadjalphvf*(1./alph2v);
    betav = 0 + edgeadjbetavf*(beta2v./alph2v);
    alphvprev = -lambda + edgeadjalphvf*(1./alphprev2v);
    betavprev = 0 + edgeadjbetavf*(betaprev2v./alphprev2v);

    s_hat = betav./alphv;
    s_hatprev = betavprev./alphvprev;
    conv(iter) = conv(iter) + norm(s_hat-s_hatprev)^2/n;
    mse(iter) = mse(iter) + norm(s_hat-s0)^2/n;
    end
end
conv= conv/mc;
mse = mse/mc;
data = [sigma2' conv mse];
save('data.dat', 'data', '-ASCII');
```