

# STAT 375 Homework 5 Solutions

## Problem (1)

One can rewrite the cost function in the following fashion:

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

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:

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

## 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

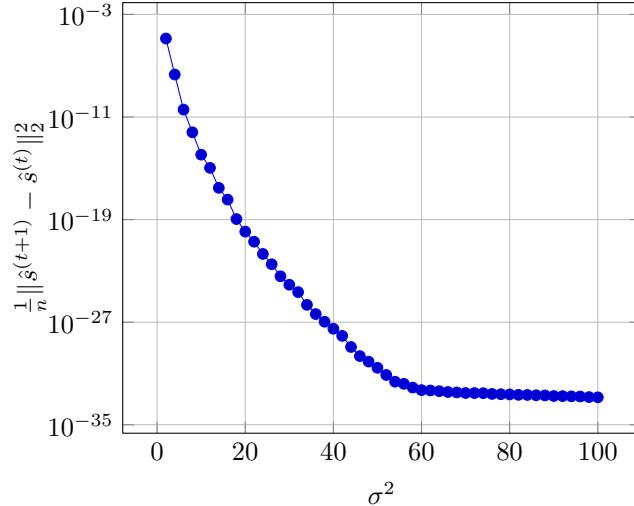


Figure 1: Convergence of the BP algorithm for different values of  $\sigma^2$

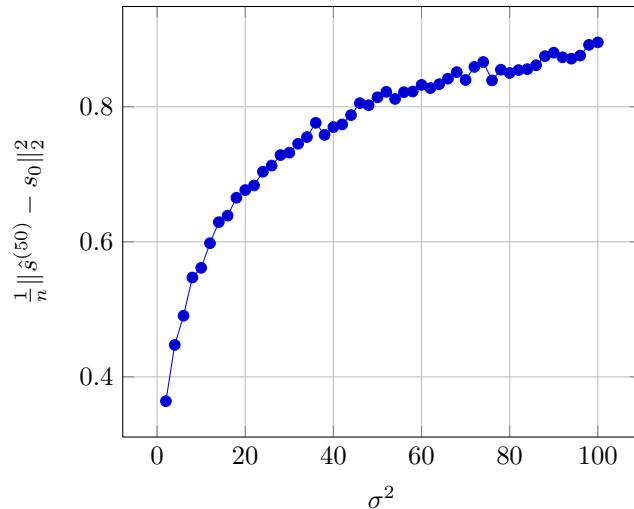


Figure 2: The mean square error of the (approximate) ridge regression estimator obtained via BP

as follows:

```

1 clear all
2 m = 800;
3 n = 1000;
4
5 l = 20;
6 i = kron(1:m, ones(1, 1));
7 j = zeros(1, length(i));

```

```

8  for iter = 1:m
9      y = randsample(n, 1)';
10     j(((iter-1)*l+1):(iter-1)*l+l) = y;
11 end
12
13 k = 2*double(rand(1, length(i)) >= 0.5) -1;
14
15 A = sparse(i, j, k, m, n);
16 [i, j, k] = find(A);
17 nz = nnz(A);
18
19
20 % Q = [zeros(m,m), A; A' zeros(n, n)];
21 % Q = sparse(Q);
22
23 % [iq, jq, kq] = find(Q);
24 % nz = nnz(Q);
25
26 uniqid = (m+n)*i + j; %This gives iq,jq uniquely
27 id = [1:length(uniqid)];
28
29 mapobj = containers.Map(uniqid,id);
30
31 iedgeadjc = zeros(1000000,1);
32 jedgeadjc = zeros(1000000,1);
33 iedgeadjv = zeros(1000000,1);
34 jedgeadjv = zeros(1000000,1);
35
36 entryedgeadjbetac =zeros(1e6,1);
37 entryedgeadjbetav = zeros(1e6,1);
38 %Edge adjacency for bp updates
39 cnt = 0;
40 cnt1 = 0;
41 for ctr=1:m
42     ctr
43     q1 = find(A(ctr,:));
44     for ctrl= q1
45         edgeid = mapobj([(m+n)*ctr+ctrl]);
46         othernodes = setdiff(q1,[ctrl]);
47         cnt1=cnt1+1;
48         for ctr2 = othernodes
49             edgeconn = mapobj([(m+n)*ctr+ctr2]);
50             cnt=cnt+1;
51             iedgeadjc(cnt)=edgeid;
52             jedgeadjc(cnt)=edgeconn;
53             entryedgeadjbetac(cnt) = -A(ctr,ctr2);
54             %entryedgeadjbeta(cnt) = -Q(ctr2,ctr);
55         end
56     end
57 end
58 edgeadjalphc = sparse(iedgeadjc(1:cnt),jedgeadjc(1:cnt),-1);
59 edgeadjbetac = sparse(iedgeadjc(1:cnt),jedgeadjc(1:cnt),entryedgeadjbetac(1:cnt));
60
61
62 cnt = 0;
63 cnt1 = 0;
64 for ctr=1:n
65     ctr
66     q1 = find(A(:,ctr));
67     for ctrl = q1'
68         %[ctrl,ctr, (m+n)*ctrl+ctr]
69         edgeid = mapobj([(m+n)*ctrl+ctr]);
70         othernodes = setdiff(q1,[ctrl]);
71         cnt1=cnt1+1;
72         for ctr2 = othernodes'
73             edgeconn = mapobj([(m+n)*ctr2+ctr]);
74             cnt=cnt+1;
75             iedgeadjv(cnt)=edgeid;
76             jedgeadjv(cnt)=edgeconn;
77             entryedgeadjbetav(cnt) = -A(ctr2,ctr);

```

```

78         %entryedgeadjbeta(cnt) = -Q(ctr2,ctr);
79     end
80 end
81
82 edgeadjalphv = sparse(iedgeadjv(1:cnt),jedgeadjv(1:cnt),-1);
83 edgeadjbetav = sparse(iedgeadjv(1:cnt),jedgeadjv(1:cnt),entryedgeadjbetav(1:cnt));
84
85 %Edge adjacency for final node expressions
86
87 iedgeadjcf = zeros(1000000,1);
88 jedgeadjcf = zeros(1000000,1);
89 iedgeadjvf = zeros(1000000,1);
90 jedgeadjvf = zeros(1000000,1);
91
92 entryedgeadjbetacf =zeros(1e6,1);
93 entryedgeadjbetavf = zeros(1e6,1);
94 %Edge adjacency for bp updates
95 cnt = 0;
96 cnt1 = 0;
97 for ctr=1:m
98     ctr
99     q1 = find(A(ctr,:));
100
101    for ctr2 = q1
102        edgeconn = mapobj([(m+n)*ctr+ctr2]);
103        cnt=cnt+1;
104        iedgeadjcf(cnt)=ctr;
105        jedgeadjcf(cnt)=edgeconn;
106        entryedgeadjbetacf(cnt) = -A(ctr,ctr2);
107        %entryedgeadjbeta(cnt) = -Q(ctr2,ctr);
108    end
109 end
110
111 edgeadjalphcf = sparse(iedgeadjcf(1:cnt),jedgeadjcf(1:cnt),-1);
112 edgeadjbetacf = sparse(iedgeadjcf(1:cnt),jedgeadjcf(1:cnt),entryedgeadjbetacf(1:cnt));
113
114
115 cnt = 0;
116 cnt1 = 0;
117 for ctr=1:n
118     ctr
119     q1 = find(A(:,ctr));
120
121    for ctr2 = q1'
122        edgeconn = mapobj([(m+n)*ctr2+ctr]);
123        cnt=cnt+1;
124        iedgeadjvf(cnt)=ctr;
125        jedgeadjvf(cnt)=edgeconn;
126        entryedgeadjbetavf(cnt) = -A(ctr2,ctr);
127        %entryedgeadjbeta(cnt) = -Q(ctr2,ctr);
128    end
129 end
130
131 edgeadjalphvf = sparse(iedgeadjvf(1:cnt),jedgeadjvf(1:cnt),-1);
132 edgeadjbetavf = sparse(iedgeadjvf(1:cnt),jedgeadjvf(1:cnt),entryedgeadjbetavf(1:cnt));
133 save('decoder2.mat');
134 load('decoder2.mat');
135
136 %now generate the samples
137 sigma2 = 2:2:100;
138 conv = zeros(length(sigma2), 1);
139 mse = conv;
140 mc = 20;
141 for iter = 1:length(sigma2)
142     iter
143     for iter2 = 1:mc
144         sigma = sqrt(sigma2(iter));
145         lambda = sigma2(iter);
146         s0 = randn(n, 1);
147         w = sigma*randn(m, 1);

```

```

148     y = A*s0+w;
149     out = y(i);
150     %now do the iterations
151     alph2c = ones(16000,1);
152     beta2c = zeros(16000,1);
153
154     alph2v = ones(16000,1);
155     beta2v = zeros(16000,1);
156
157
158    for ctr=1:50
159        alphprev2c = alph2c;
160        betaprev2c = beta2c;
161        alphprev2v = alph2v;
162        betaprev2v = beta2v;
163
164        alph2v = 1+edgeadjalphc*(1./alphprev2c);
165        beta2v = out + edgeadjbetac*(betaprev2c./alphprev2c);
166        alph2c = -lambda + edgeadjalphv*(1./alphprev2v);
167        beta2c = 0 + edgeadjbetav*(betaprev2v./alphprev2v);
168
169    end
170
171    %Compute final estimates
172    solridge = (A'*A+lambda*eye(n))\ (A'*y);
173    % % % norm(solridge-bpestimatemean(m+1:n+m))
174
175    alphv = -lambda + edgeadjalphvf*(1./alph2v);
176    betav = 0 + edgeadjbetavf*(beta2v./alph2v);
177    alphvprev = -lambda + edgeadjalphvf*(1./alphprev2v);
178    betavprev = 0 + edgeadjbetavf*(betaprev2v./alphprev2v);
179
180    s_hat = betav./alphv;
181    s_hatprev = betavprev./alphvprev;
182    conv(iter) = conv(iter) + norm(s_hat-s_hatprev)^2/n;
183    mse(iter) = mse(iter) + norm(s_hat-s0)^2/n;
184
185    end
186 end
187 conv= conv/mc;
188 mse = mse/mc;
189 data = [sigma2' conv mse];
190 save('data.dat', 'data', '-ASCII');

```