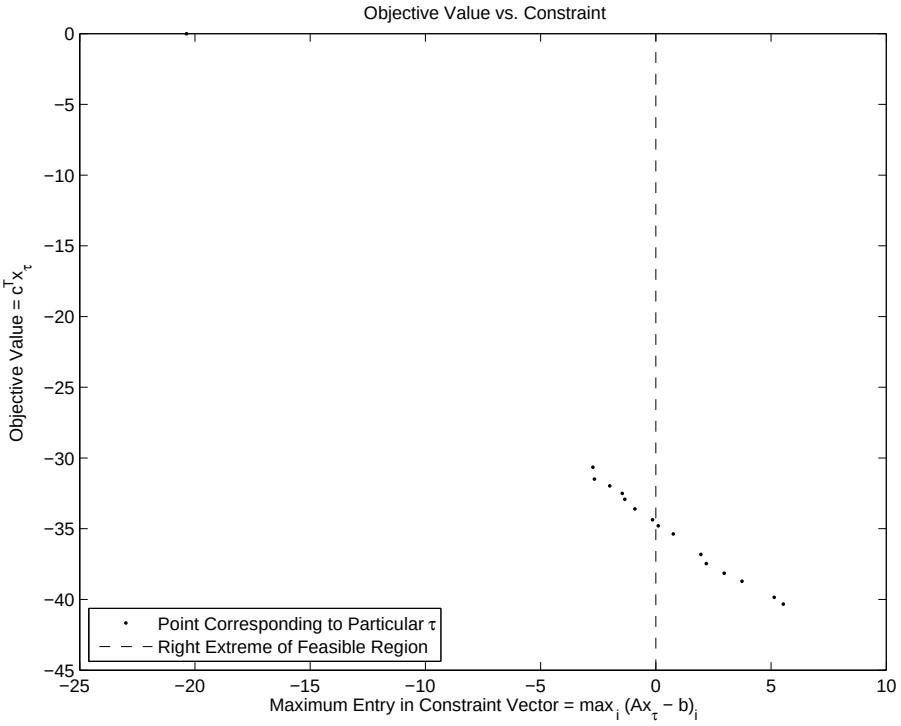
1. After randomly generating A and $\mathbf{c}$, setting $\mathbf{b} = \frac{1}{2}A\mathbf{1}$, and solving the continuous relaxation of the boolean linear programing problem, I got a $\mathbf{y}^*$. It only had 16 entries with values that weren't 0 or 1. $\mathbf{c}^T\mathbf{y}^* = -35.396$, which we know is a lower bound for $\mathbf{c}^T\mathbf{x}$, the optimal objective value for the original problem. I generated $\mathbf{x}_\tau$ for $\tau = .01n, n = 1$ to 100, and plotted the objective value against the minimum slack among the rows of $A\mathbf{x}_\tau \leq \mathbf{y}$ below.



The feasible points lie to the left of 0, corresponding to $A\mathbf{x}_\tau - \mathbf{y}$ having all negative entries (and thus the constraint being satisfied). Since each point in this region is a feasible point for the original boolean program, the corresponding objective value is an upper bound for the minimum objective value, since the minimum must be less than or equal to each of them. The smallest of these values is $\mathbf{c}^T\mathbf{x} = -34.373$. Thus, we have an upper bound on the objective value at optimal point of -34.373 and a lower bound of -35.396, the difference being 1.023.

Note: Since there are only 16 values in $\mathbf{y}$ which aren't either 0 or 1, there could be at most unique 17 $\mathbf{x}_\tau$ (there are actually only 16 since two of the points which aren't 0 or 1 are within .01 of each other). This is because $\mathbf{x}_\tau = \mathbf{x}_{\tau+.01}$ if none of the entries of $\mathbf{y}$ lie in $(\tau, \tau + .01)$.

Code:

```
% set seed and generate matricies/vectors
rand('state',0)
A = rand(300,100)
c = rand(100,1)-1
b = .5 * A * ones(100,1)

% solve the linear programing problem to get optimal y and lower bound l
cvx_begin
    variable y(100)
```

```
    minimize(c'*y)
    subject to
        A*y <= b
        0 <= y <= 1
cvx_end
l = cvx_optval

% For values of tau going from 0 to 1 in increments of .01, round values
% of y above tau to 0 and above tau to 1 to get xtau (I store all 100 vectors
% in one matrix). Then, calculate objective value of c'x for each tau
tau = .01*(1:100)                    % tau comes out as a 1x100 matrix
% xtau(r,c) is 1 if y(r)>=tau(c), and otherwise 0
xtau = repmat(y,1,100) >= repmat(tau,100,1)
obj = c' * xtau

% Test inequality constraint: get the largest entry value of A*x -b for each
con = max(A * xtau - repmat(b,1,100),[],1)

% Pick the best objective value for which the constraint is satistifed and
% compare to our initial lower bound
% each entry of obj is negative, so zeroing out some entries doesn't matter
u = min(obj .* (con<=0))
diff = u - l

% Plot objective value vs constraint
plot(con,obj,'.k')
hold on
plot([0,0],[-45,0],'--r')
title('Objective Value vs. Constraint')
xlabel('Maximum Entry in Constraint Vector = max _i (Ax_\tau - b)_i')
ylabel('Objective Value = c^Tx_\tau')
legend('Point Corresponding to Particular \tau','Right Extreme of Feasible Re
,'location','southwest')
print('plot_1.pdf')
hold off
close
```

2. If we set $A = \mathbf{a}\mathbf{a}'$, this problem can be equivalently written as, where $X \in \mathbb{S}^n$,

$$\begin{aligned} \text{minimize} \quad & tr(AX) \\ \text{subject to} \quad & X \succeq 0 \\ & X = \begin{bmatrix} .2 & x_1 & x_2 & x_3 \\ x_1 & .1 & x_4 & x_5 \\ x_2 & x_4 & .3 & x_6 \\ x_3 & x_5 & x_6 & .1 \end{bmatrix} \\ & x_1 \geq 0 \qquad\qquad\qquad\qquad x_2 \geq 0 \\ & x_4 \leq 0 \qquad\qquad\qquad\qquad x_5 \leq 0 \\ & x_6 \geq 0 \end{aligned}$$

Plugging this into our software, we get that

$$X^* = \begin{bmatrix} .2 & 0 & .2449 & 0 \\ 0 & .1 & 0 & -.1 \\ .2449 & 0 & .3 & 0 \\ 0 & -.1 & 0 & .1 \end{bmatrix}, \; tr(AX^*) = .0013$$

Its interesting to note that our minimal objective value, even with the restrictions placed on X, is quite close to the minimum value you would get for arbitrary $X \succeq 0$ of 0.

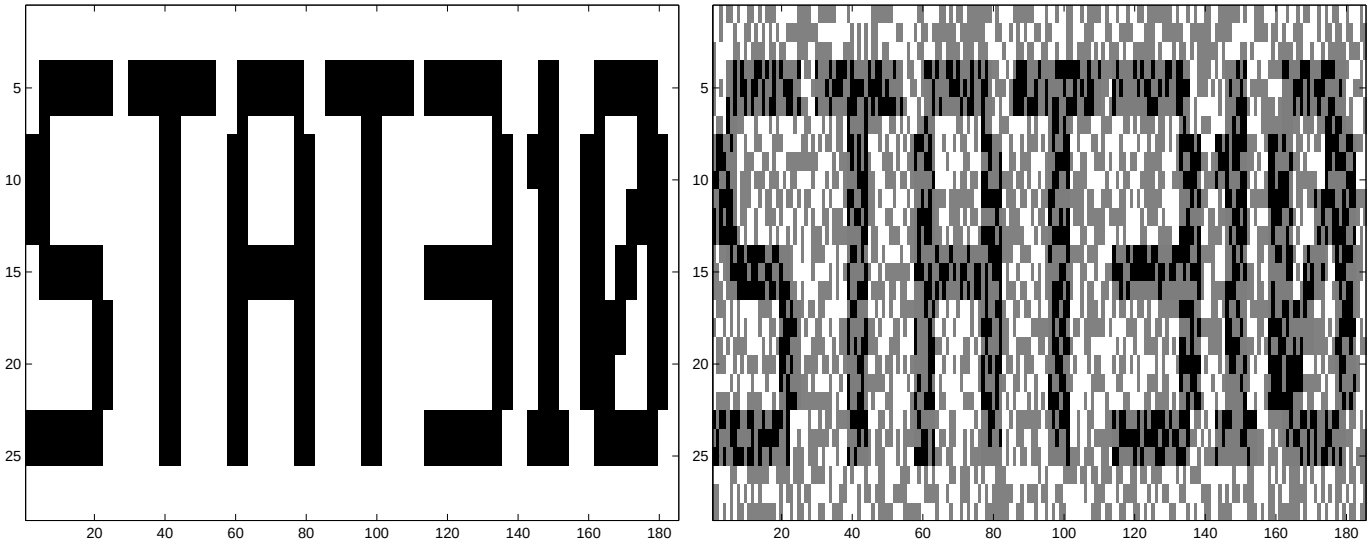
Code:

```
% input matricies
a = [.1;.2;-.05;.1]
A = a * a'
b = [.2;.1;.3;.1]    % known diagonal of X

% Optimize cvx. The objective is a'Xa = trace(AX), and the constraints specify
% X is PSD and has the appropriate entries.
cvx_begin
    variable X(4,4) symmetric
    minimize(trace(A*X))
    subject to
        X == semidefinite(4)
        diag(X) == b
        X(1,2) >= 0
        X(1,3) >= 0
        X(2,3) <= 0
        X(2,4) <= 0
        X(3,4) >= 0
cvx_end
```

3. We can transform X into a vector $\mathbf{x}$ such that $x_{ij} = \mathbf{x}_{ni+j}$ to make this closer to the perfered form for a LP/QP problem. Then, we can make a sparse matrix C such that $C\mathbf{x} = \nabla X$. Each row of C will have all 0s except one 1 and one -1. In the first $(m-1)n$ rows of C, the -1 will be along the main diagonal, and the 1 will be n spots to the right, making $\mathbf{c}_i'\mathbf{x} = x_ij - x_{i-1,j}$. In the remaining $m(n-1)$ rows of C, there will be -1s along the diagonal starting from the top left corner of the section and 1s one spot to the right, making $\mathbf{c}_i'\mathbf{x} = x_ij - x_{i,j-1}$.

   Now, here is the original image on the left, and the original image with the missing pixes greyed out on the right:



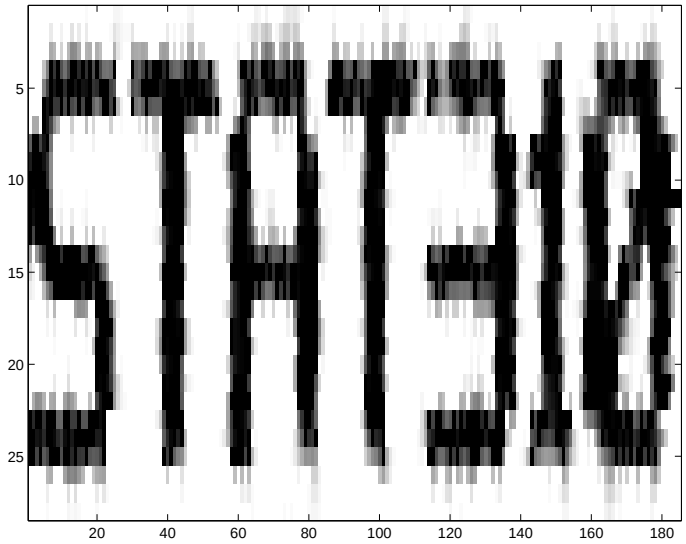
(a) When we use the L_2-variation, our problem can be formatted for CVX as:

$$\begin{aligned} &\text{minimize} && \|C\mathbf{x}\|_2 \\ &\text{subject to} && x_{ij} = a_{ij},\ (i,j) \in S \end{aligned}$$

Which has the equivalent solution to the QP problem

$$\begin{aligned} &\text{minimize} && \mathbf{x}'C'C\mathbf{x} \\ &\text{subject to} && x_{ij} = a_{ij},\ (i,j) \in S \end{aligned}$$

Optimizing the first problem in CVX, and transforming $\mathbf{x}^*$ into X^*, we get this image:



It's decently close to the original, but blurry and not sharp at all

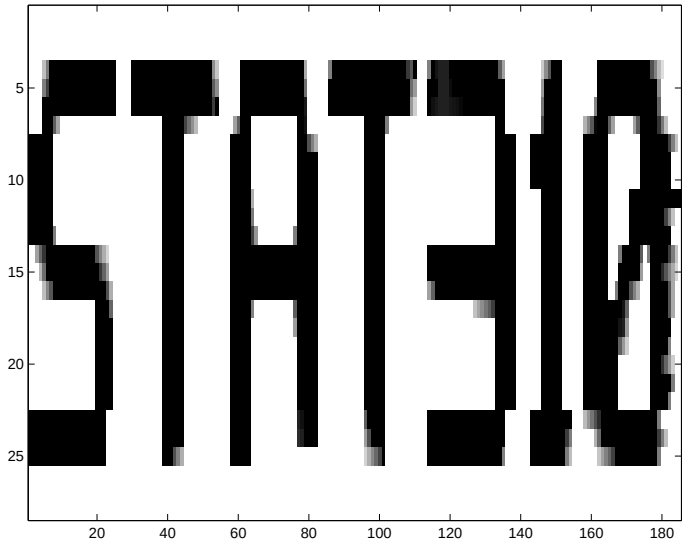
(b) When we use the L_1-variation, our problem can be formatted for CVX as:

$$\begin{aligned} &\text{minimize} && \|C\mathbf{x}\|_1 \\ &\text{subject to} && x_{ij} = a_{ij},\ (i,j) \in S \end{aligned}$$

Which has an equivalent solution to the LP problem

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}'\mathbf{t} \\ \text{subject to} \quad & \mathbf{t} \geq C\mathbf{x} \\ & \mathbf{t} \geq -C\mathbf{x} \\ & x_{ij} = a_{ij},\ (i,j) \in S \end{aligned}$$

Optimizing the first problem in CVX, and transforming $\mathbf{x}^*$ into X^*, we get this image:



Which is much better; its much closer to the original and doesn't have nearly as much blurring as with the L_2 variation.

Code:

```
% Get Matricies.
newimage          % Given program

% Convert matricies so this can be formatted as a QP/LP. Matricies huge but s
% For both problems, the objective is a norm of Cx
xs = sparse(reshape(S.*X,[m*n,1]))       % elements of X, or 0 if data missing
SD = sparse(diag(reshape(S,[m*n,1])))    % has elements of S along diagonal
CT = speye([m*(n-1),m*n]) - [sparse(m*(n-1),m),speye(m*(n-1),m*(n-1))] % Top
CB = speye([(m-1)*n,m*n]) - [sparse((m-1)*n,1),speye((m-1)*n,m*n-1)] % bottom
C  = [CT;CB]

% Show original image
colormap gray
imagesc(X)
print('image_2_orig.pdf')
close

% Show sparse image
colormap gray
imagesc(abs(X-127*(1-S)))
print('image_2_miss.pdf')
close
```

```
%% part a: L2 variation.

% optimize the problem, in this case a QP since were minimizing the 2 norm
% SD*x == xs is asserting x_ij = a_ij when S=1, and is 0=0 when S=0 (always t
cvx_begin
    variable x(m*n)
    minimize(norm(C*x,2))
    subject to
        SD*x == xs
cvx_end

% output the image
colormap gray
imagesc(reshape(x,[m,n]))
print('image_2a.pdf')
close
cvx_optval_a = cvx_optval

%% part b: L1 variation.

% optimize the problem, in this case a LP
cvx_begin
    variable x(m*n)
    minimize(norm(C*x,1))
    subject to
        SD*x == xs
cvx_end
cvx_optval_b = cvx_optval

% output the image
colormap gray
imagesc(reshape(x,[m,n]))
print('image_2b.pdf')
```

4. (a) A simple way to estimate x with a LCQP is to minimize the squared error over the first k terms where y is known under the restriction that the predictions for the remaining terms are higher than β. If $A_1 = [\mathbf{a}_1, ..., \mathbf{a}_k]', A_2 = [\mathbf{a}_{k+1}, ..., \mathbf{a}_n]'$, and $\mathbf{b}_1 = [b_1, ..., b_k]$, this gives us the optimization problem

$$\begin{aligned} \text{minimize} \quad & \|A_1\mathbf{x} - b_1\|_2^2 \\ \text{subject to} \quad & A_2\mathbf{x} \geq \beta\mathbf{1} \end{aligned}$$

This is a simplification which will probably bias our prediction upwards. Just because an observed value is higher than β doesn't mean its expectation is, since the error could be greater than $\mathbf{a}_i'\mathbf{x} - \beta$. If we knew something about the distribution of the error, we could, for instance, instead have the constraint $A_2 x \geq c\mathbf{1}$, where c is a constant such that $P(\max_{k<i\leq n}(\varepsilon_i) \geq \beta - c) \leq \alpha$. This would give us $1 - \alpha$ confidence that we are allowing for the true values of $A_2\mathbf{x}$.

We don't have enough information to do this in the data description though, so instead I ran the first version I gave, giving

$$\mathbf{x}_{qp} = \begin{matrix}[-0.22 & -1.69 & 0.40 & 0.18 & -1.06 & 1.28 & 1.13 & -0.01 & 0.30 & 0.38 \\ -0.14 & 0.81 & -0.32 & 2.12 & -0.24 & -0.02 & 0.97 & -0.11 & -0.15 & -0.89]\end{matrix}$$

(b) The usual least squares estimator based on just the first k points is given by

$$\begin{aligned}\mathbf{x}_{ls} &= (A_1'A_1)^{-1}A_1'b_1 \\ &= \begin{matrix}[-0.35 & -1.80 & 0.20 & 0.17 & -0.84 & 1.30 & 1.83 & -0.56 & 0.37 & -0.05 \\ -0.11 & 1.53 & -0.50 & 2.42 & -0.56 & -0.37 & 0.99 & -0.25 & -0.18 & -0.43]\end{matrix}\end{aligned}$$

(c) Comparing these two estimates to the original, the LCQP has the smaller relative error:

$$\frac{\|\mathbf{x}_{qp} - \mathbf{x}\|_2}{\|\mathbf{x}\|_2} = .1538$$

while

$$\frac{\|\mathbf{x}_{ls} - \mathbf{x}\|_2}{\|\mathbf{x}\|_2} = .3907$$

Code:

```
% get the data.
censoring              % given program
AL = A(:,1:k)          % left 25 columns A
AR = A(:,(k+1):n)      % remainder of A

%% part a: fit a constrained QP.
% Usual least squares for the first k data points,
% constraint that a'x>beta for remainder
cvx_begin
    variable x_qp(d)
    minimize(norm(AL'*x_qp-b,2))
    subject to
        AR'*x_qp>=beta
cvx_end

%% part b: usual least squares
x_ls = (AL*AL')^-1 * AL*b

%% part c: comparison
re_qp = norm(x_qp - x_true)/norm(x_true)
```

5. The problem as given is

$$\begin{aligned}&\text{minimize} && \|\mathbf{x}\|_1 \\ &\text{subject to} && \mathrm{E}[(y - \mathbf{a}'\mathbf{x})^2] \leq .01\mathrm{E}(y^2)\end{aligned}$$

Using a few identities from statistics, we can transform the constraint into a quadratic constraint:

$$\begin{aligned}\mathrm{E}[(y - \mathbf{a}'\mathbf{x})^2] &\leq .01\mathrm{E}[y^2] \\ (\mathrm{E}[y] - \mathrm{E}[\mathbf{a}'\mathbf{x}])^2 + \mathrm{Var}(y - \mathbf{a}'\mathbf{x}) &\leq .01(\mathrm{E}[y]^2 + \mathrm{Var}(y)) \\ (\mu'\mathbf{c} - \mu'\mathbf{x})^2 + \mathrm{Var}(y) - 2\mathrm{Cov}(y, \mathbf{a}'\mathbf{x}) + \mathrm{Var}(\mathbf{a}'\mathbf{x}) &\leq .01((\mu'\mathbf{c})^2 + \mathrm{Var}(y)) \\ (\mu'\mathbf{c} - \mu'\mathbf{x})^2 + \mathbf{c}'\Sigma\mathbf{c} - 2\mathbf{c}'\Sigma\mathbf{x} + \mathbf{x}'\Sigma\mathbf{x} &\leq .01((\mu'\mathbf{c})^2 + \mathbf{c}'\Sigma\mathbf{c}) \\ .99(\mu'\mathbf{c})^2 - 2\mu'\mathbf{c}\mu'\mathbf{x} + (\mu'\mathbf{x})^2 + .99\mathbf{c}'\Sigma\mathbf{c} - 2\mathbf{c}'\Sigma\mathbf{x} + \mathbf{x}'\Sigma\mathbf{x} &\leq 0\end{aligned}$$

Giving us the QCLP

$$\begin{aligned} &\text{minimize} && \|\mathbf{x}\|_1 \\ &\text{subject to} && .99(\mu'\mathbf{c})^2 - 2\mu'\mathbf{c}\mu'\mathbf{x} + (\mu'\mathbf{x})^2 + .99\mathbf{c}'\Sigma\mathbf{c} - 2\mathbf{c}'\Sigma\mathbf{x} + \mathbf{x}'\Sigma\mathbf{x} \leq 0 \end{aligned}$$

Our optimal $\mathbf{x}^*$ isn't perfectly sparse, but its quite close, with the majority of its entries close to 0. Of the 500 entries, only 50 are larger than .1, 52 are larger than .001, 69 are larger than .0001, and 373 are larger than .00001.

Code:

```
% get the data
sparsify          % given program
Ey = c'*mu        % expectation of c'y
Vy = c'*Sigma*c % variance of c'y
cS = c'*Sigma   % when multiplied by x gives cov(c'a,y'a)

% Run the QCLP
% we are minimizing L1 of x. The constraints are turned into QC using the ide
% E(y)^2=E(y^2)+var(y) and MSE(y-hy) = E(y-hy)^2 + var(y-hy)
cvx_begin
    variable x(n)
    minimize(norm(x,1))
    subject to
        .99*Ey^2 - 2*Ey*x'*mu + (x'*mu)^2 + .99*Vy - 2*cS*x + x'*Sigma*x<=0
cvx_end

% now, calculate the number of entries of x above various thresholds:
xpos=zeros(6,1)
for i=1:6
    xpos(i)=sum(x>10^-i)
end
```