CS/ECE/ME 532
Homework 5: The SVD and Least Squares

1. Recall the face emotion classification problem from HW 2. Design and compare the performances of the classifiers proposed in **a** and **b**, below. In each case, divide the dataset into 8 equal sized subsets (e.g., examples $1-16$, $17-32$, etc). Use 6 sets of the data to estimate $\boldsymbol{x}$ for each choice of the *regularization parameter*, select the best value for the regularization parameter by estimating the error on one of the two remaining sets of data, and finally use the $\boldsymbol{x}$ corresponding to the best value of the regularization parameter to predict the labels of the remaining "hold-out" set. Compute the number of mistakes made on this hold-out set and divide that number by 16 (the size of the set) to estimate the error rate. Repeat this process 56 times (for the 8×7 different choices of the sets used to select the regularization parameter and estimate the error rate) and average the error rates to obtain a final estimate.

a. Truncated SVD solution. Use the pseudo-inverse $\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T$, where $\boldsymbol{\Sigma}^{-1}$ is computed by inverting the k largest singular values and setting others to zero. Here, k is the regularization parameter and it takes values $k = 1, 2, \ldots, 9$; i.e., compute 9 different solutions, $\widehat{\boldsymbol{x}}_k$.

b. Regularized LS. Let $\widehat{\boldsymbol{x}}_\lambda = \arg\max_{\boldsymbol{x}} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_2^2$, for the following values of the regularization parameter $\lambda = 0, 2^{-1}, 2^0, 2^1, 2^2, 2^3$, and 2^4. Show that $\widehat{\boldsymbol{x}}_\lambda$ can be computed using the SVD and use this fact in your code.

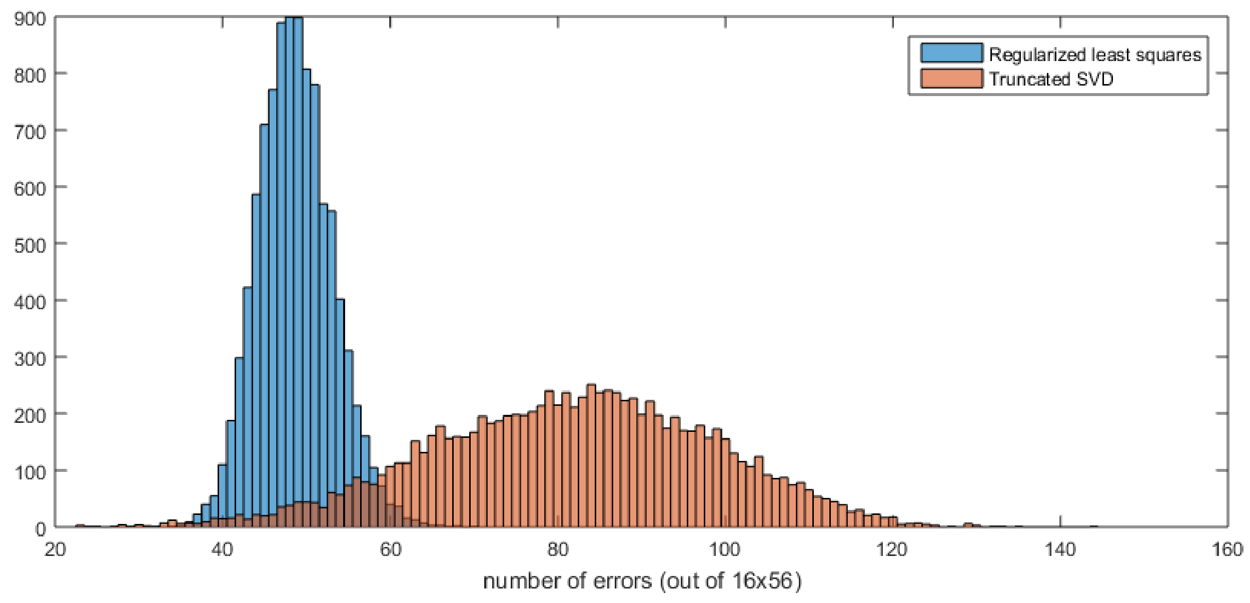
SOLUTION: Running the code below, we find an average error rate of `0.1116` for the truncated SVD and an average error rate of `0.0480` for regularized least-squares (RLS). So RLS appears to have the lowest error

```
load face_emotion_data
[~,m] = size(X);

error_RLS = zeros(8,7);  % error rate for regularized least squares
error_SVD = zeros(8,7);  % error rate for svd truncation

% SVD parameters to test
kvals = 1:9;
err_mini_SVD = zeros(numel(kvals),1);

% RLS parameters to test
lamvals = 2.^[-inf -1:4];
err_mini_RLS = zeros(numel(lamvals),1);

% LOOP OVER FIRST HOLD-OUT SET
for h = 1:8
   ih = 16*h-15:16*h;              % the set of hold-out indices
   it = setdiff(1:8*16,ih);        % the set of training indices

    % first holdout set and training set
    Xhh = X(ih,:); yhh = y(ih);
    Xtt = X(it,:); ytt = y(it);

    % LOOP OVER SECOND HOLD-OUT SET
    for j = 1:7
        jh = 16*j-15:16*j;          % inner set of hold-out indices
        jt = setdiff(1:7*16,jh);    % inner set of training indices

        % second holdout set and training set
        Xh = Xtt(jh,:); yh = ytt(jh);
        Xt = Xtt(jt,:); yt = ytt(jt);
        [U,S,V] = svd(Xt,'econ');  % u will be skinny; s and v square

        % TEST SVD TRUNCATION
        wt = zeros(m,numel(kvals));
        for i = 1:numel(kvals)
            k = kvals(i);
            % compute estimate by using only first k singular vectors
            wt(:,i) = V(:,1:k)*diag( 1./diag(S(1:k,1:k)) )*(U(:,1:k)'*yt);
            yp = sign(Xh*wt(:,i));   % error on 2nd holdout
            err_mini_SVD(i) = mean( yp ~= yh );
        end
        [~,ind] = min(err_mini_SVD); % choose best based on 2nd holdout
        ypp = sign(Xhh*wt(:,ind));   % compute error based on 1st holdout
        error_SVD(h,j) = mean( ypp ~= yhh );

        % TEST REGULARIZED LEAST-SQUARES
        wt = zeros(m,numel(lamvals));
        for i = 1:numel(lamvals)
            lambda = lamvals(i);
            % compute estimate by using regularization
            wt(:,i) = V*diag( diag(S)./(diag(S).^2 + lambda) )*(U'*yt);
            yp = sign(Xh*wt(:,i));   % error on 2nd holdout
            err_mini_RLS(i) = mean( yp ~= yh );
        end
        [~,ind] = min(err_mini_RLS); % choose best based on 2nd holdout
        ypp = sign(Xhh*wt(:,ind));   % compute error based on 1st holdout
        error_RLS(h,j) = mean( ypp ~= yhh );
    end
end

% compute average error rate over all trials
avg_err_rate_SVD = mean(error_SVD(:))
avg_err_rate_RLS = mean(error_RLS(:))
```

c. Use the original dataset to generate 3 new features for each face, as follows. Take the 3 new features to be random linear combination of the original 9 features. This can be done with the Matlab command `A*randn(9,3)` and augmenting the original matrix $\boldsymbol{A}$ with the resulting 3 columns. Will these new features be helpful for classification? Why or why not? Repeat the experiments in (a) and (b) above using the 12 features.

SOLUTION: The code is very similar to the code above so we omit it. This time, we add three random features (extra columns) to A that are linear combinations of the existing columns. The plot below shows the error distribution over 10,000 trials. The average error rates over all these trials is 0.0911 $\pm$ 0.0188 for the truncated SVD and 0.0543 $\pm$ 0.0050 for RLS. So RLS still outperforms SVD in this case.

It is not surprising that adding random features does not improve the classifier. The new random columns are linear combinations of the existing columns, so range(A) does not change. There is still variability however, because of (1) the nonlinearity inherent in our classification method (the `sign` function), (2) the fact that $\|x\|_2$ can actually be reduced when we add more columns (even though $\|Ax - b\|_2$ can't be improved), so the λ values we test get shifted and the weighting changes.



2. Many sensing and imaging systems produce signals that may be slightly distorted or blurred (e.g., an out-of-focus camera). In such situations, algorithms are needed to deblur the data to obtain a more accurate estimate of the true signal. The Matlab code `blurring.m` generates a random signal and a blurred and noisy version of it, similar to the example shown below. The code simulates this equation:

$$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{e}\,,$$

where $\boldsymbol{b}$ is the blurred and noisy signal, $\boldsymbol{A}$ is a matrix that performs the blurring operation, $\boldsymbol{x}$ is the true signal, and $\boldsymbol{e}$ is a vector of errors/noise. The goal is to estimate $\boldsymbol{x}$ using $\boldsymbol{b}$ and $\boldsymbol{A}$.

a. Implement the standard LS, truncated SVD, and regularized LS methods for this problem.

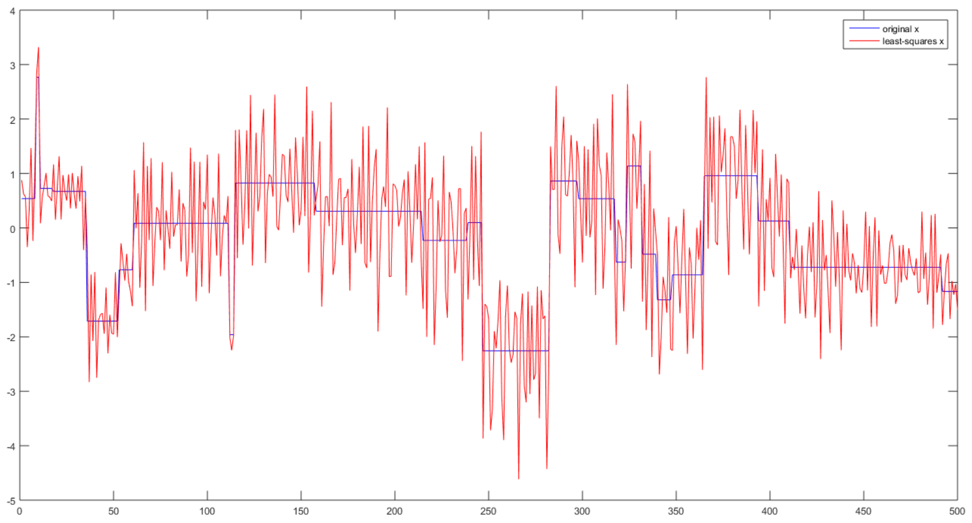
SOLUTION: One easy way to implement each estimator is to start with $\boldsymbol{A}$ and $\boldsymbol{b}$ and compute the following for example:

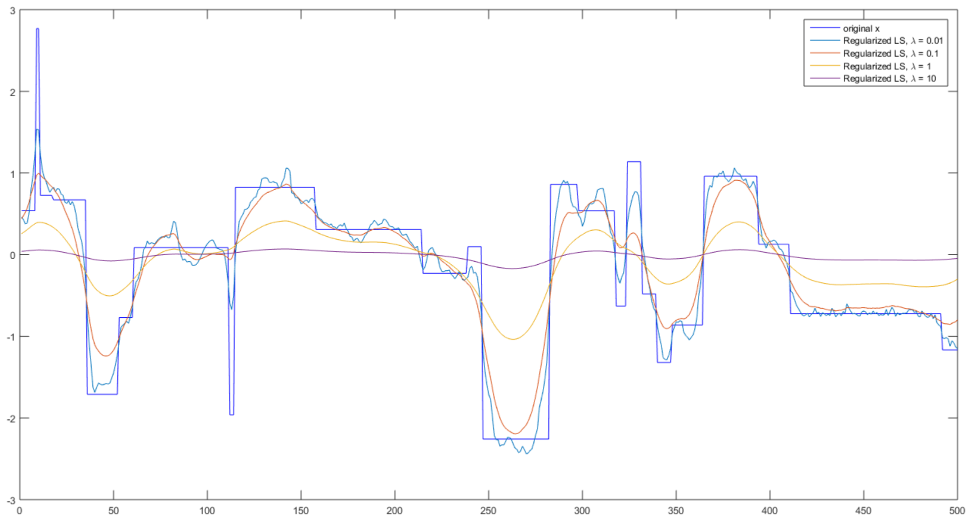
```
[U,S,V] = svd(A,'econ');

x_LS = V*diag( diag(S).^(-1) )*(U'*b);

% (loop over k values)
x_SVD = V(:,1:k)*diag( diag(S(1:k,1:k)).^(-1) )*(U(:,1:k)'*b);

% (loop over L (lambda) values)
x_RLS = V*diag( diag(S).*( (diag(S).^2+L).^(-1) ) )*(U'*b);
```

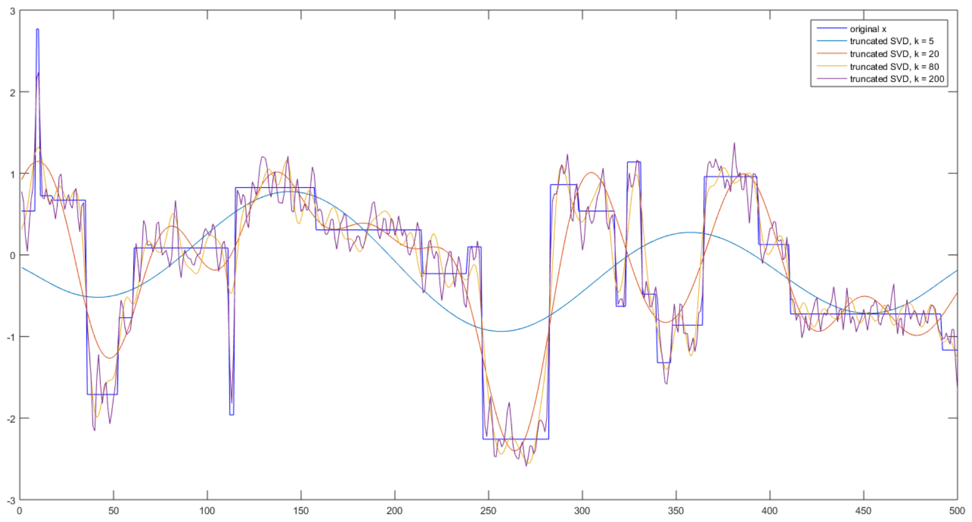
Note that I used `diag( diag(S).^(-1) )` rather than `inv(S)` since we know S is diagonal—it's more efficient to take the diagonal elements, invert them individually, and then rebuild a diagonal matrix.

b. Experiment with different averaging functions (i.e., different values of k in the code) and with different noise levels (σ in the code). How do the blurring and noise level affect the value of the regularization parameters that produce the best estimates?

SOLUTION: See the following page for sample estimators with default noise:


original x
least-squares x


original x
Regularized LS, λ = 0.01
Regularized LS, λ = 0.1
Regularized LS, λ = 1
Regularized LS, λ = 10


original x
truncated SVD, k = 5
truncated SVD, k = 20
truncated SVD, k = 80
truncated SVD, k = 200

A great deal can be said about these estimators. Here are some examples of observations.

- The LS estimator is very noisy. This is because every singular value of A gets inverted. The smallest singular values are quite small, so when they get inverted they become large. These large values multiply the noisy $\boldsymbol{b}$ vector and ultimately amplify the noise. Even though the noise on $\boldsymbol{b}$ is relatively small, it gets amplified and this is evident in the estimate $\widehat{\boldsymbol{x}}$.
- The RLS estimator adds a regularization *before* inverting. So rather than computing $1/\sigma$ for each singular value of A, we compute $\sigma/(\sigma^2+\lambda)$, which is smaller than $1/\sigma$. This moves σ away from zero before inverting, which causes the noise to be amplified less. However, as λ gets larger the inverse is driven to zero. In the plot, we see that the limit $\lambda \to 0$ recovers the noisy LS estimate, and as λ gets larger the estimates are simultaneously de-noised and driven towards zero.
- The SVD estimator simply truncates the small singular values rather than inverting them. So as in the RLS case we are reducing the effect of noise, but without driving the estimate to zero. The result is akin to a Fourier series approximation, except the basis used is not the Fourier basis—it is a basis determined by the singular vectors. An important twist is that there is an additional trade-off: including more singular values increases the quality of the estimate, but also increases susceptibility to noise.