CS/ECE/ME 532
Homework 3

1. Orthogonal columns. Consider the matrix and vector

$$\boldsymbol{A} = \begin{bmatrix} 3 & 1 \\ 0 & 3 \\ 0 & 4 \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}.$$

a) By hand, find two orthonormal vectors that span the plane spanned by columns of $\boldsymbol{A}$.

SOLUTION: We can begin by normalizing the first vector. This gives us:

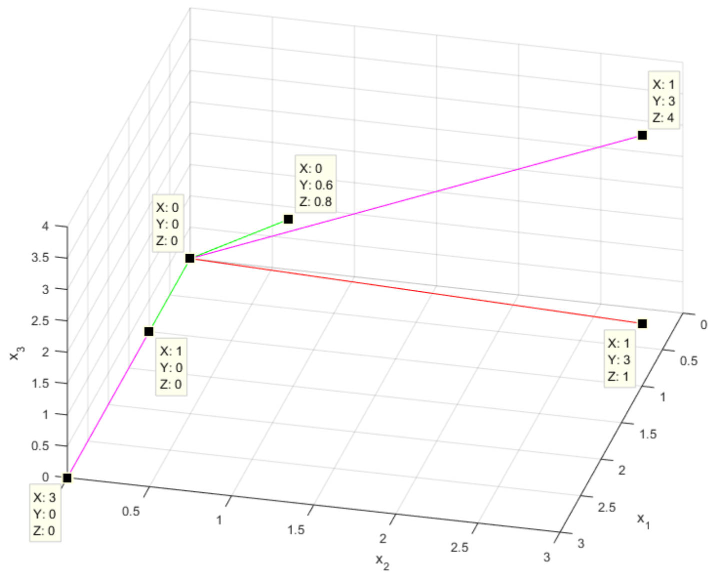
$$\boldsymbol{u}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

To find the second vector, we need a vector that is orthogonal to $\boldsymbol{u}_1$ and in the span of the columns of $\boldsymbol{A}$. One way to ensure orthogonality in this case is to cancel out the first component of the second column of $\boldsymbol{A}$. So subtract $\boldsymbol{u}_1$ from $\boldsymbol{a}_{\cdot 2}$, and normalize:

$$\boldsymbol{u}_2 = \begin{bmatrix} 0 \\ \frac{3}{5} \\ \frac{4}{5} \end{bmatrix}$$

b) Make a sketch of these vectors and the columns of $\boldsymbol{A}$ in three dimensions.

SOLUTION: Here is a sketch. Again, we will accept anything that is roughly correct. $\boldsymbol{A}$-vectors in magenta, $\boldsymbol{u}$-vectors in green, $\boldsymbol{b}$-vector in red.



c) Use these vectors to compute the LS estimate $\hat{\boldsymbol{b}} = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}$.

SOLUTION: knowing the orthogonalization of $\boldsymbol{A}$ makes the computation much easier! The least-squares solution $\hat{\boldsymbol{b}}$ will be the same if we replace $\boldsymbol{A}$ by $\boldsymbol{U}$. Then, we have:

$$\begin{aligned}
\hat{\boldsymbol{b}} &= \boldsymbol{U}(\boldsymbol{U}^T\boldsymbol{U})^{-1}\boldsymbol{U}^T\boldsymbol{b} \\
&= \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{b} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & \frac{3}{5} \\ 0 & \frac{4}{5} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \frac{3}{5} \\ 0 & \frac{4}{5} \end{bmatrix}^T \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & \frac{3}{5} \\ 0 & \frac{4}{5} \end{bmatrix} \begin{bmatrix} 1 \\ \frac{13}{5} \end{bmatrix} \\
&= \begin{bmatrix} 1 \\ \frac{39}{25} \\ \frac{52}{25} \end{bmatrix}
\end{aligned}$$

Where we used the fact that $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$ because $\boldsymbol{U}$ is orthogonal. We can also check that the answer is the same using $\boldsymbol{A}$ (but more work):

$$\begin{aligned}
\hat{\boldsymbol{b}} &= \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b} \\
&= \begin{bmatrix} 3 & 1 \\ 0 & 3 \\ 0 & 4 \end{bmatrix} \left(\begin{bmatrix} 3 & 1 \\ 0 & 3 \\ 0 & 4 \end{bmatrix}^T \begin{bmatrix} 3 & 1 \\ 0 & 3 \\ 0 & 4 \end{bmatrix} \right)^{-1} \begin{bmatrix} 3 & 1 \\ 0 & 3 \\ 0 & 4 \end{bmatrix}^T \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 3 & 1 \\ 0 & 3 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 9 & 3 \\ 3 & 26 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 14 \end{bmatrix} \\
&= \frac{1}{225} \begin{bmatrix} 3 & 1 \\ 0 & 3 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 26 & -3 \\ -3 & 9 \end{bmatrix} \begin{bmatrix} 3 \\ 14 \end{bmatrix} \\
&= \frac{1}{225} \begin{bmatrix} 3 & 1 \\ 0 & 3 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 36 \\ 117 \end{bmatrix} \\
&= \frac{1}{225} \begin{bmatrix} 225 \\ 351 \\ 468 \end{bmatrix} \\
&= \begin{bmatrix} 1 \\ \frac{39}{25} \\ \frac{52}{25} \end{bmatrix}
\end{aligned}$$

2. Gram-Schmidt. Write your own code to perform Gram-Schmidt orthogonalization. Your code should take as input a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and return as output a matrix $\boldsymbol{U} \in \mathbb{R}^{m\times r}$ where $\boldsymbol{U}$ is orthogonal and has the same range as $\boldsymbol{A}$. Note that r will indicate the rank of $\boldsymbol{A}$, so your code can also be used to find the rank of a matrix!

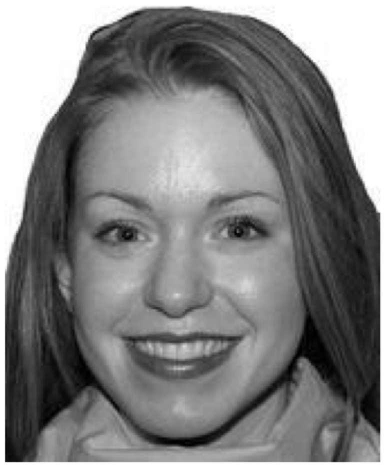
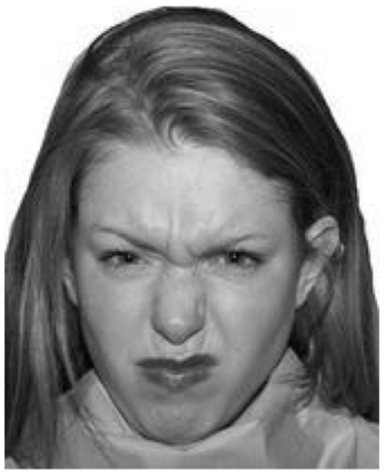
SOLUTION: Here is a (spectacularly short) solution in Matlab:

```
function U = gram_schmidt( A )
   % GRAM-SCHMIDT CODE
   %  U = gram_schmidt( A )
   %  where U'*U=I and range(U)=range(A)
   %  number of columns of U is the rank of A.

   [m,n] = size(A);
   U = zeros(m,0);           % start with empty matrix
   for i = 1:n
      v = A(:,i);            % the current column of A
      v = v - U*(U'*v);      % project onto current output set
      if norm(v) > 1e-12     % ensure linear independence
         U = [U v/norm(v)]; % normalize and add to the set
      end
   end
end  % end of function
```

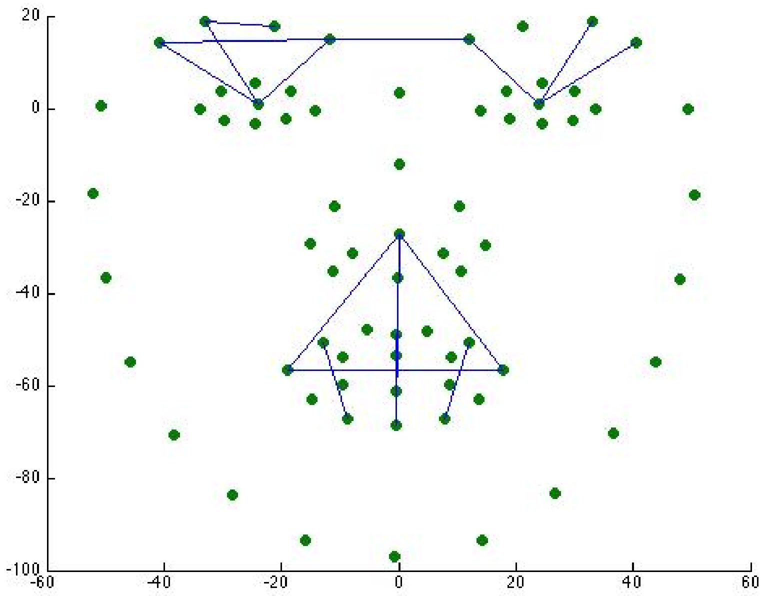
3. Design classifier to detect if a face image is happy.

Consider the two faces below. It is easy for a human, like yourself, to decide which is happy and which is not. Can we get a machine to do it?

The key to this classification task is to find good features that may help to discriminate between happy and mad faces. What features do we pay attention to? The eyes, the mouth, maybe the brow?

The image below depicts a set of points or "landmarks" that can be automatically detected in a face image (notice there are points corresponding to the eyes, the brows, the nose, and the mouth). The distances between pairs of these points can indicate the facial expression, such as a smile or a frown. We chose $n = 9$ of these distances as features for a classification algorithm. The features extracted from $m = 128$ face images (like the two shown above) are stored in the $m \times n$ matrix $\mathbf{A}$ in the Matlab file `face_emotion_data.mat`. This file also includes an $m \times 1$ binary vector $\mathbf{b}$; happy faces are labeled $+1$ and mad faces are labeled -1. The goal is to find a set of weights for the features in order to predict whether the emotion of a face image is happy or mad.



a) Use the training data $\mathbf{X}$ and $\mathbf{y}$ to find an good set of weights.

SOLUTION: Loading the data into matlab and solving the least-squares problem:

```
% load a and b vectors
load polydata.mat

m = numel(a);                       % number of data points
N = 100;                            % num points to use for interpolation
z = linspace(min(a),max(a),N);      % pts where interpolant is evaluated
y = zeros(3,N);                     % where we'll store polynomial values

for d = 1:3
```

```
    % generate A-matrix for this choice of d
    A = zeros(m,d+1);
    for i = 1:m
       for j = 1:d+1
          A(i,j) = a(i)^(j-1);
       end
    end

    % solve least-squares problem. x is the list of coefficients.
    % NOTE: a shortcut in matlab is to just type: x = A\b;
    x = inv(A'*A)*(A'*b);

    % evaluate best-fit polynomial at all points z. store result in y.
    % NOTE: you can do this in one line with the polyval command!
    for i = 1:N
       for j = 1:d+1
          y(d,i) = y(d,i) + x(j)*z(i)^(j-1);
       end
    end

end

% plot the data and the best-fit polynomials
figure(1)
plot(a,b,'.', z,y(1,:), z,y(2,:), z,y(3,:),'LineWidth',2)
legend('data','d=1','d=2','d=3','Location','NorthWest')
title('best-fit␣polynomials␣of␣degree␣1,2,3')
```

The result is:

$$w = \begin{bmatrix} 0.9437 \\ 0.2137 \\ 0.2664 \\ -0.3922 \\ -0.0054 \\ -0.0176 \\ -0.1663 \\ -0.0823 \\ -0.1664 \end{bmatrix}$$

b) How would you use these weights to classify a new face image as happy or mad?

SOLUTION: Here is the process:

i. extract the 9 features from the new face image. Place them in a row vector v^T.

ii. compute the product $s = v^T w$, where w is the weight vector calculated in part a.

iii. If $s > 0$ (closer to $+1$), classify as *happy*. Otherwise (closer to -1), classify as *mad*. Applying a cutoff like this is called *thresholding*.

c) Which features seem to be most important? Justify your answer.

SOLUTION: We will argue that weights with larger absolute value are associated with features that are more important. Remember that the classifier computes a weighted combination of the features. If the features are $v_1, \ldots, v_9$, then

$$s = w_1 v_1 + \ldots w_9 v_9$$

Moreover, the features are normalized in this example—each column of X has a squared norm of 128. Therefore, if a weight is small, then the relative contribution of that feature to the total s will be commensurately small.

d) Can you design a classifier based on just 3 of the 9 features? Which 3 would you choose? How would you build a classifier?

SOLUTION: To build a classifier based on just 3 of the 9 features, I would choose the three features with the largest associated weights. These turn out to be features 1, 3, and 4. To build the classifier, we remove the columns of X corresponding to the features we are no longer using. Call this new matrix Z. Then we solve the exact same least-squares problem as before; $Zw = b$.

e) A common method for estimating the performance of a classifier is cross-validation (CV). CV works like this. Divide the dataset into 8 equal sized subsets (e.g., examples $1 - 16$, $17 - 32$, etc). Use 7 sets of the data to chose your weights, then use the weights 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 8 times (for the 8 different choices of the hold-out set) and average the error rates to obtain a final estimate.

SOLUTION: Here is Matlab code that computes cross-validation:

```
load face_emotion_data

err_rate = zeros(8,1);  % will store error rates for each test

% loop over possible hold-out sets
for k = 1:8
   ih = [16*k-15:16*k];     % the set of hold-out indices
   it = setdiff(1:128,ih); % the set of training indices

   % solve least-squares problem using training data
   Xt = X(it,:);
   yt = y(it,:);
   wt = inv(Xt'*Xt)*(Xt'*yt);
   % predict labels of hold-out faces with weights from training set
   Xh = X(ih,:);
   yh = y(ih,:);
   yp = sign(Xh*wt);  % +1 if >0 and -1 if <0.

   % compute error rate for this experiment
   err_rate(k) = mean( yp ~= yh );
end

% compute average error rate
avg_err_rate = mean(err_rate)
```

The output of the code above is that the average error rate is 0.0469. Therefore, the average error rate using cross-validation is 4.69%.

f) What is the estimated error rate using all 9 features? What is it using the 3 features you chose in (d) above?

SOLUTION: Here is Matlab code that computes the error rates for the full feature set and also for only the top three features.

```
load face_emotion_data

% compute weights using all the data
w = inv(X'*X)*(X'*y);
y_predicted  = sign(X*w);
err_rate = mean( y ~= y_predicted )

% compute weights using only features 1,3,4.
Xr = X(:,[1 3 4]);
wr = inv(Xr'*Xr)*(Xr'*y);
yr_predicted = sign(Xr*wr);
err_rate_reduced = mean( y ~= yr_predicted )
```

The result is an error rate of 2.34% using all the features, and 6.25% using only features 1,3,4.