ECE/CS 532
Homework 2: Least Squares

1. Answer the following questions. Make sure to explain your reasoning.

a. Are the columns of the following matrix linearly independent?

$$\boldsymbol{A} = \begin{bmatrix} +0.5 & +0.5 \\ -0.5 & +0.5 \\ +0.5 & -0.5 \\ -0.5 & -0.5 \end{bmatrix}$$

SOLUTION: Yes. Since the first elements match, the only way one column can be a multiple of the other is if all elements match, which is not the case.

b. Are the columns of the following matrix linearly independent?

$$\boldsymbol{A} = \begin{bmatrix} +1 & +1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & -1 \end{bmatrix}$$

SOLUTION: Yes. This can be shown by bringing the matrix to reduced row echelon form:

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix} \xrightarrow[r_2 + r_3 \mapsto r_3]{} \begin{bmatrix} 1 & 1 & 1 \\ -1 & 1 & -1 \\ 0 & 0 & -2 \end{bmatrix} \xrightarrow[r_1 + r_2 \mapsto r_2]{} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & -2 \end{bmatrix} \xrightarrow[r_1 - \frac{1}{2}r_2 + \frac{1}{2}r_3 \mapsto r_1]{} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -2 \end{bmatrix}$$

The final matrix is diagonal with nonzero entries so the matrix is full-rank (and therefore invertible in this case, since it's also square).

c. What is the rank of the following matrix?

$$\boldsymbol{A} = \begin{bmatrix} +2 & +1 \\ -2 & +1 \\ +2 & -1 \end{bmatrix}$$

SOLUTION: The rank is at most 2 because there are 2 columns. The columns are not multiples of one another so they are independent and therefore $\operatorname{rank}(A) = 2$.

d. Suppose the matrix in part c is used in the least squares optimization $\min_{\boldsymbol{x}} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$. Does a unique solution exist?

SOLUTION: The normal equations are $A^T A x = A^T b$. And any solution to the normal equations is a solution (minimizer) of the least-squares problem. Because A has full column rank, $A^T A$ is invertible, and so the normal equations have a unique solution. Consequently the least-squares problem also has a unique solution.

2. Consider the following matrix and vector:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}.$$

a. Find the solution $\widehat{\boldsymbol{x}}$ to $\min_{\boldsymbol{x}} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$.

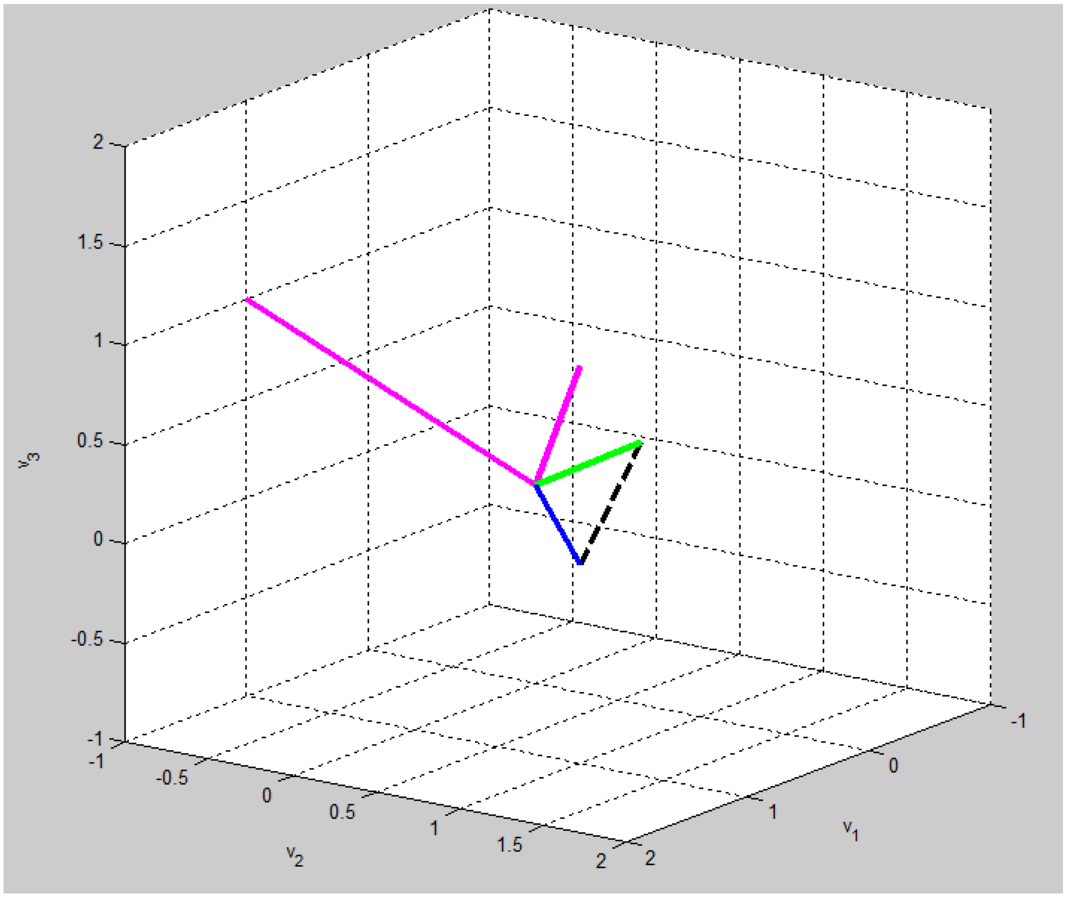
SOLUTION: The normal equations are: $A^T A x = A^T b$. Substituting A and b, they become:

$$\begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

From the second equation, $x_1 = -3x_2$. Substituting into the first equation, $-8x_2 = 2$. Therefore, $x_1 = \frac{3}{4}$ and $x_2 = -\frac{1}{4}$.

b. Make a sketch of the geometry of this particular problem in $\mathbb{R}^3$, showing the columns of $\boldsymbol{A}$, the plane they span, the target vector $\boldsymbol{b}$, the residual vector and the solution $\widehat{\boldsymbol{b}} = \boldsymbol{A}\widehat{\boldsymbol{x}}$.

SOLUTION: ok... so this was tricky to draw so I cheated and did it in matlab. We'll accept anything remotely close as a valid sketch!



The magenta lines are the columns of A, the blue line is b, and the green line is the least-squares solution (which lies in the plane spanned by the magenta vectors). The dashed line is the path of the projection (also the residual).

3. Polynomial fitting. Suppose we observe pairs of points (a_i, b_i), $i = 1, \dots, m$. Imagine these points are measurements from a scientific experiment. The variables a_i are the experimental conditions and the b_i correspond to the measured response in each condition. Suppose we wish to fit a degree $d < m$ polynomial to these data. In other words, we want to find the coefficients of a degree d polynomial p so that $p(a_i) \approx b_i$ for $i = 1, 2, \dots, m$. We will set this up as a least-squares problem.

a. Suppose p is a degree d polynomial. Write the general expression for $p(a_i) = b$.

SOLUTION: A polynomial of degree d has the form: $p(z) = x_0 + x_1 z + x_2 z^2 + \cdots + x_d z^d$. Where $x_0, \dots, x_d$ are the coefficients. Note that there are $d+1$ coefficients. Therefore, the equation $p(a_i) = b_i$ is:

$$x_0 + x_1 a_i + x_2 a_i^2 + \cdots + x_d a_i^d = b_i$$

b. Express the $i = 1, \dots, m$ equations as a system in matrix form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Specifically, what is the form/structure of $\boldsymbol{A}$ in terms of the given a_i.

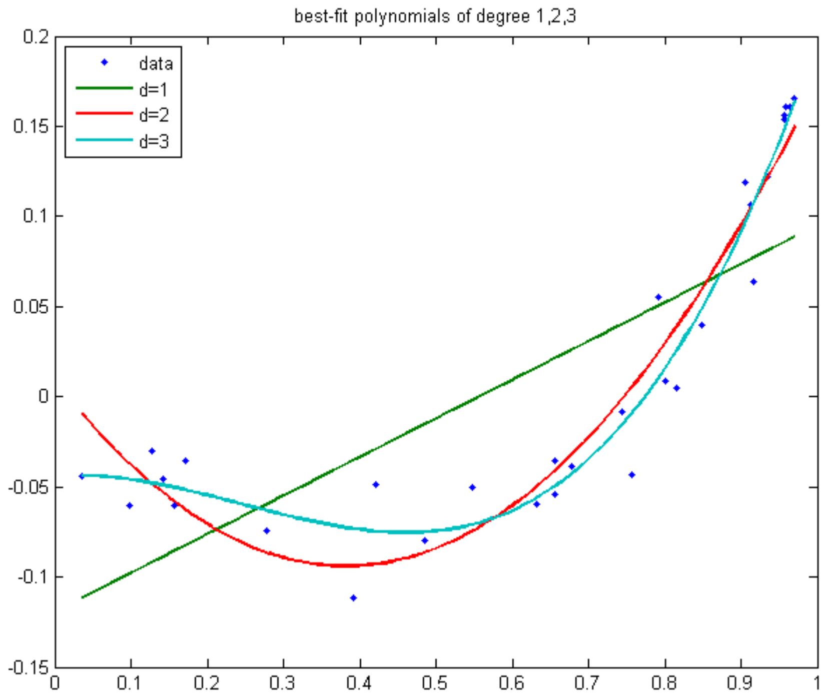
SOLUTION: Writing out the equations from the previous part for $i = 1, \dots, m$ and stacking them into a matrix, we obtain the equation:

$$\begin{bmatrix} 1 & a_1 & a_1^2 & \dots & a_1^p \\ 1 & a_2 & a_2^2 & \dots & a_2^p \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & a_m & a_m^2 & \dots & a_m^p \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{bmatrix}$$

Interesting bit of trivia: matrices of this form are known as *Vandermonde* matrices.

c. Write a Matlab or Python script to find the least-squares fit to the $m = 30$ data points in `polydata.mat`. Plot the points and the polynomial fits for $d = 1, 2, 3$.

SOLUTION: Here is the output of the code:



And here is the Matlab code that produced the plot.

```
% 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')
```

4. Recall the cereal calorie prediction problem discussed in class. The data matrix for this problem is

$$\boldsymbol{A} = \begin{bmatrix} 25 & 0 & 1 \\ 20 & 1 & 2 \\ 40 & 1 & 6 \end{bmatrix}.$$

Each row contains the grams/serving of carbohydrates, fat, and protein, and each row corresponds to a different cereal (*Frosted Flakes, Froot Loops, Grape-Nuts*). The total calories for each cereal are

$$\boldsymbol{b} = \begin{bmatrix} 110 \\ 110 \\ 210 \end{bmatrix}.$$

a. Write a small program (e.g., in Matlab or Python) that solves the system of equations $\boldsymbol{Ax} = \boldsymbol{b}$. Recall the solution $\boldsymbol{x}$ gives the calories/gram of carbohydrate, fat, or protein. What is the solution?

SOLUTION: The solution is: $x = \begin{bmatrix} 4.2500 \\ 17.5000 \\ 3.7500 \end{bmatrix}$. Matlab code that produces it is:

```
A = [ 25 0 1
      20 1 2
      40 1 6 ];
b = [ 110 110 210 ]';
x = inv(A)*b
```

b. The solution may not agree with the known calories/gram, which are 4 for carbs, 9 for fat and 4 for protein. We suspect this may be due to rounding the grams to integers, especially the fat grams. Assuming the true value for calories/gram is

$$\boldsymbol{x}^\star = \begin{bmatrix} 4 \\ 9 \\ 4 \end{bmatrix},$$

and that the total calories, grams of carbs, and grams of protein are correctly reported above, determine the "correct" grams of fat in each cereal.

SOLUTION: Let the unknown grams of fat be f_1, f_2, f_3. The new equations are:

$$\begin{bmatrix} 25 & f_1 & 1 \\ 20 & f_2 & 2 \\ 40 & f_3 & 6 \end{bmatrix} \begin{bmatrix} 4 \\ 9 \\ 4 \end{bmatrix} = \begin{bmatrix} 110 \\ 110 \\ 210 \end{bmatrix}$$

Rearranging and solving for the f's, we obtain:

$$f_1 = \frac{6}{9} \approx 0.667 \qquad f_2 = \frac{22}{9} \approx 2.444 \qquad f_3 = \frac{26}{9} \approx 2.889$$

c. Now suppose that we predict total calories using a more refined breakdown of carbohydrates, into total carbohydrates, complex carbohydrates and sugars (simple carbs). So now we will have 5 features to predict calories (the three carb features + fat and protein). So let's suppose we measure the grams of these features in 5 different cereals to obtain this data matrix

$$\boldsymbol{A} = \begin{bmatrix} 25 & 15 & 10 & 0 & 1 \\ 20 & 12 & 8 & 1 & 2 \\ 40 & 30 & 10 & 1 & 6 \\ 30 & 15 & 15 & 0 & 3 \\ 35 & 20 & 15 & 2 & 4 \end{bmatrix},$$

and the total calories in each cereal

$$\boldsymbol{b} = \begin{bmatrix} 104 \\ 97 \\ 193 \\ 132 \\ 174 \end{bmatrix}.$$

Can you solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$? Carefully examine the situation in this case. Is there a solution that agrees with the true calories/gram?

SOLUTION: Since the total carbs are equal to the complex carbs plus the sugars, the first column of the A matrix is equal to the sum of the second and third columns. So the A matrix is *not* full rank. Therefore, we can't simply invert it and solve for x. To see whether the system of equations still has a solution, we can use Matlab to check that:

$$\operatorname{rank}(A) = 4 \qquad \text{and} \qquad \operatorname{rank}(\begin{bmatrix} A & b \end{bmatrix}) = 4$$

since adding the column b to the matrix A does not increase its rank, b must lie in the span of the columns of A (i.e. there is a solution to $Ax = b$). But how do we find it? One way to do this is to remove the first column of A, which is redundant. We then have:

$$\begin{bmatrix} 15 & 10 & 0 & 1 \\ 12 & 8 & 1 & 2 \\ 30 & 10 & 1 & 6 \\ 15 & 15 & 0 & 3 \\ 20 & 15 & 2 & 4 \end{bmatrix} z = \begin{bmatrix} 104 \\ 97 \\ 193 \\ 132 \\ 174 \end{bmatrix}$$

Solving the associated least-squares problem (which is now full-rank), we obtain the solution:

$$z = \begin{bmatrix} 4 \\ 4 \\ 9 \\ 4 \end{bmatrix}$$

But this isn't just the least-squares solution... it's an exact solution to the problem! (the residual is zero). Moreover, it agrees with the true calories per gram! Due to the rank deficiency of A, the solution to the original problem is not unique. The most general solution to the original problem is:

$$x = \begin{bmatrix} \alpha \\ 4-\alpha \\ 4-\alpha \\ 9 \\ 4 \end{bmatrix} \qquad \text{for any } \alpha \in \mathbb{R}$$

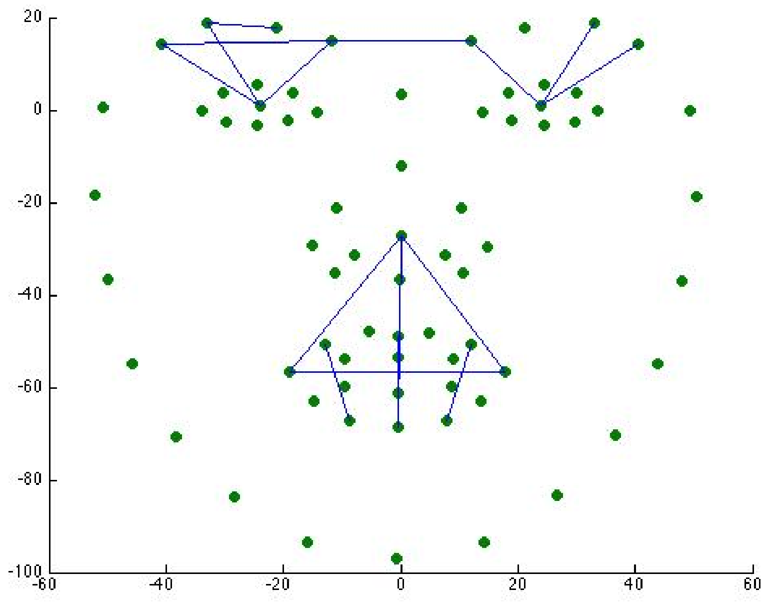
and the solution that agrees with the true carbs/gram is when $\alpha = 0$. Note that this problem is very special. Usually in such situations (e.g. if b is chosen randomly but we keep the same A matrix), there won't exist a solution to $Ax = b$ and we would have to resort to solving the least squares problem (i.e. finding an x that is "close" to being a solution).

5. **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 **A** and **b** to find an good set of weights.

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

```
load face_emotion_data

% X is 128x9. Each row is a face, each column a feature
% y is 128x1. Each row is +1 if smiling, -1 if frowning (labels)

% we assume X is full-rank here
w = inv(X'*X)*(X'*y)
```

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:

1. extract the 9 features from the new face image. Place them in a row vector v^T.
2. compute the product $s = v^T w$, where w is the weight vector calculated in part a.
3. 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 $v_1, \dots, v_9$, then

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

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.