1. $\underline{a}_1 \quad \underline{a}_2$

$$A = \begin{bmatrix} 3 & 1 \\ 0 & 3 \\ 0 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$$

(a) let $\underline{u}_1 = \dfrac{\underline{a}_1}{\|\underline{a}_1\|_2} = \dfrac{1}{\sqrt{3^2}} \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$

residual $\underline{a}_2' = \underline{a}_2 - \underline{u}_1 (\underline{u}_1^T \underline{a}_2)$

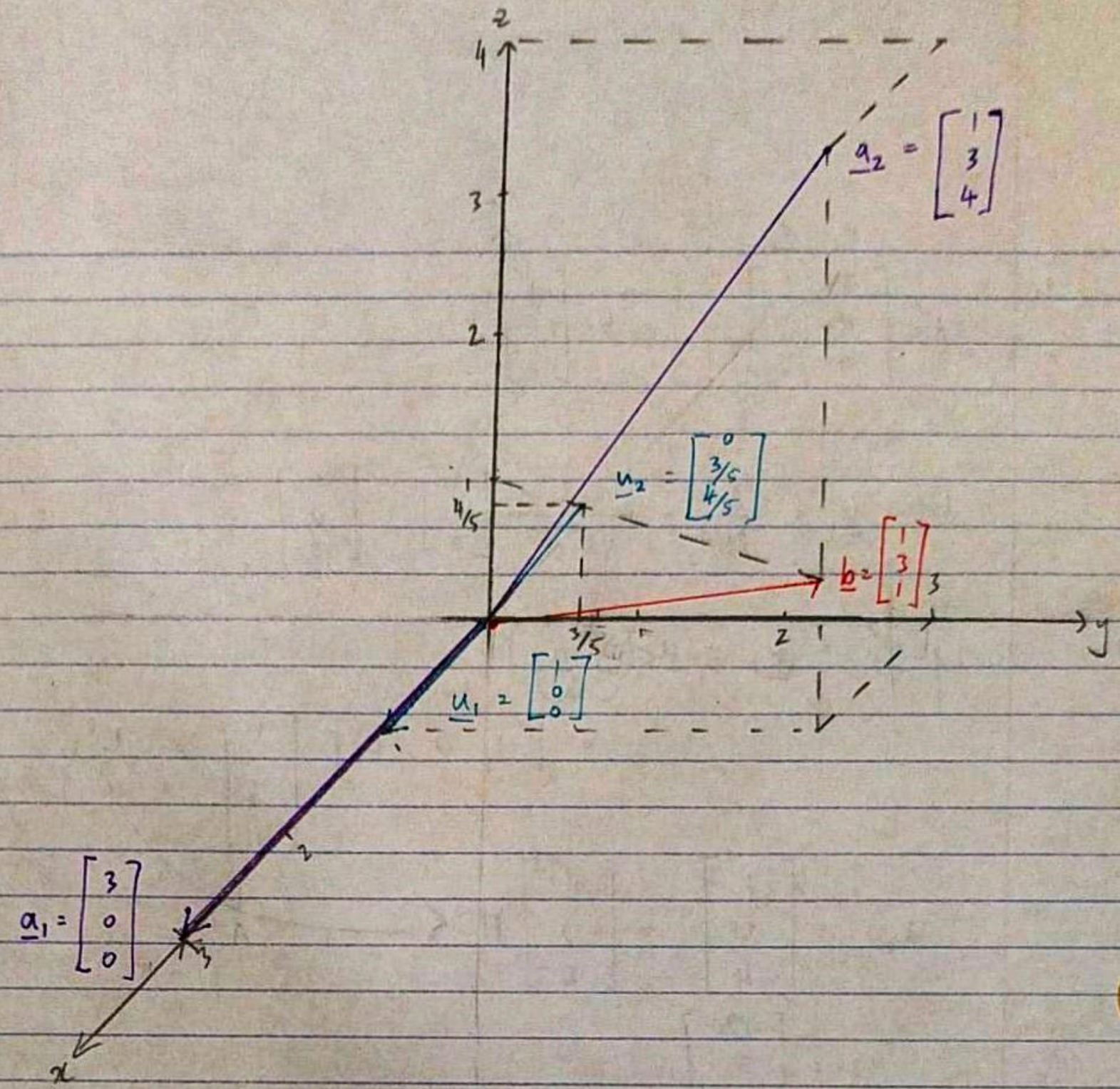
$$= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = 1$$

$$\underline{a}_2' = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} 1$$

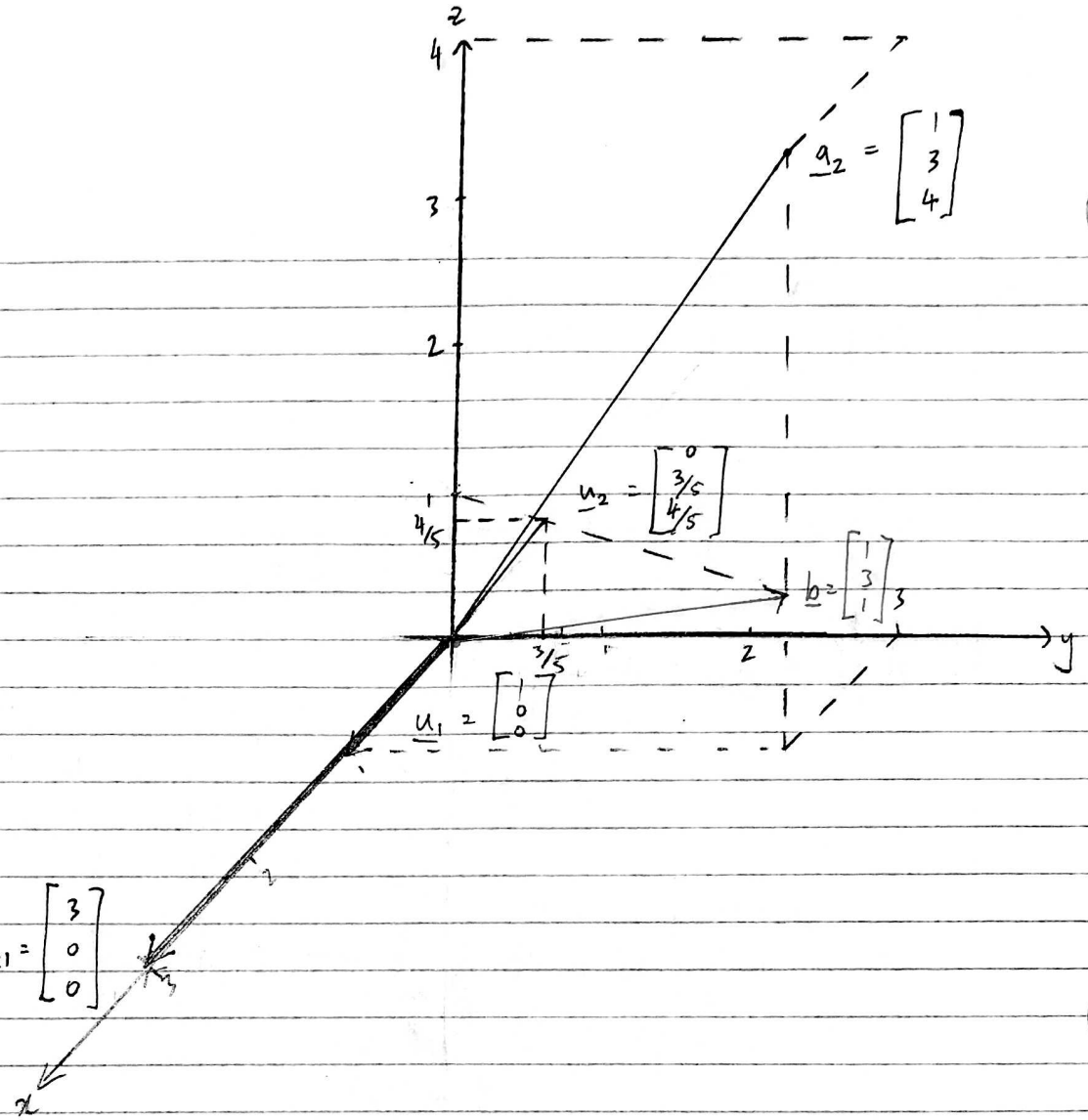
$$= \begin{bmatrix} 0 \\ 3 \\ 4 \end{bmatrix}$$

$$\underline{u}_2 = \frac{\underline{a}_2'}{\|\underline{a}_2'\|_2} = \frac{1}{\sqrt{(0)^2 + 3^2 + 4^2}} \begin{bmatrix} 0 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3/5 \\ 4/5 \end{bmatrix}$$

$$\therefore U = [\underline{u}_1, \underline{u}_2] = \begin{bmatrix} 1 & 0 \\ 0 & 3/5 \\ 0 & 4/5 \end{bmatrix}$$


2
4
3
2
4/5
3/5
2
3
y
x
$\underline{a}_2 = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$
$\underline{u}_2 = \begin{bmatrix} 0 \\ 3/5 \\ 4/5 \end{bmatrix}$
$\underline{b} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$
$\underline{u}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$
$\underline{a}_1 = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$

(b)



(c) $\hat{b} = A(A^TA)^{-1}A^Tb$

$= u\underbrace{(u^Tu)^{-1}}_{=I}u^Tb$

$= uu^Tb$

$= \begin{bmatrix} 1 & 0 \\ 0 & 3/5 \\ 0 & 4/5 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 3/5 & 4/5 \end{bmatrix}\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}$

$3\times 2 \quad 2\times 3 \quad 3\times 1$

$= \begin{bmatrix} 1 & 0 \\ 0 & 3/5 \\ 0 & 4/5 \end{bmatrix}\begin{bmatrix} 1 \\ 13/5 \end{bmatrix}$

$3\times 2 \quad 2\times 1$

$= \begin{bmatrix} 1 \\ 39/25 \\ 52/25 \end{bmatrix}$

```
function U = GS_orthogonalization(A)
% ECE 532 HW3 U2
%   U is the matrix of the Gram-Schmidt orthogonalized vectors
 spanning A's
%   vectors

[m, n] = size(A);

R = zeros(n);
U = zeros(m, n);

R(1, 1) = norm(A(:, 1));
U(:, 1) = A(:, 1)/R(1, 1);

for k = 2:n
    R(1:k-1, k) = U(:, 1:k-1)' * A(:, k);
    U(:, k) = A(:, k) - U(:, 1:k-1) * R(1:k-1, k);
    R(k, k) = norm(U(:, k));

    if R(k, k) == 0
        break;
    end

    U(:, k) = U(:, k) / R(k, k);
end
end
```

```
ans =

    1.0000         0
         0    0.6000
         0    0.8000
```

A = [3,1;0,3;0,4]
from question 1

Published with MATLAB® R2016b

Table of Contents

ECE 532 - Fall 2017 - HW3 - Q3

by Roumen Guha

```
clear all
close all

load('hw3_532_fall17_face_emotion_data.mat');
```

(a)

Use the training data X and y to find a good set of weights.

```
w_hat = X\y
```

```
w_hat =

    0.9437
    0.2137
    0.2664
   -0.3922
   -0.0054
   -0.0176
   -0.1663
   -0.0823
   -0.1664
```

(b)

How would you use these weights to classify a new face image as happy or mad? Compute the vector $\hat{y}$. Classify values that are > 0 as happy, and values that are < 0 as mad. For values that are equal to 0, we flip a coin. But that seems unlikely with floating point numbers.

```
classify = @(x) sum([1*(x>0), -1*(x<0)], 2);                    %
 Classify elements in our label y_hat

y_hat = X*w_hat
classify(y_hat)
```

```
y_hat =

   -1.2213
    0.9028
   -0.8457
    1.4290
    1.1719
   -0.7742
    0.5086
   -0.4010
    0.5778
    0.4054
   -0.9929
    0.3190
    0.5130
    0.4769
    0.1388
   -0.8354
   -1.0016
   -1.0185
    0.0182
   -0.5263
    0.9823
   -1.0297
    0.7901
   -1.0635
   -1.1497
    1.0657
   -0.3932
    1.4694
   -0.7945
    0.6237
   -1.0983
    0.7137
   -0.6957
    0.5637
   -1.3049
    0.4560
   -0.5629
    1.2538
   -0.0629
    0.3256
   -0.7738
   -0.1501
    0.3044
   -1.4096
    0.7789
   -0.0903
   -0.4948
    0.9766
    1.4176
   -0.4158
    0.2467
```

```
-0.7686
 0.6176
-1.2054
-0.7052
 1.4440
-1.0197
 1.3158
-1.1850
 0.4077
-0.8142
 0.1168
-0.9825
-1.6438
 0.8017
-1.1624
-1.3387
-0.7885
 0.5606
 1.3401
-0.1741
 0.7998
-0.7235
 1.1888
-0.6382
 1.9878
-0.6818
 0.6314
 0.1495
 0.9378
 0.6272
-1.3775
-0.1686
-1.0956
 0.9487
-0.7790
 0.6588
-0.9243
 1.3350
 0.4792
-1.0941
 0.7613
-0.9744
-0.7718
-1.3318
 0.5802
-0.8308
-1.0354
 0.5267
-0.1280
 0.9708
-0.6410
 0.8610
 1.0368
-0.4353
```

```
    1.0039
   -0.9109
    0.8886
    1.0972
   -0.7632
    0.8296
   -1.4434
    0.6699
   -0.7308
    0.3168
   -1.4137
    0.1557
    0.8668
   -0.7632
   -0.6467
    1.1023
    0.7206
    1.2416
   -0.3895
    0.6370
    0.2048
    0.7689
    1.5691


ans =

    -1
     1
    -1
     1
     1
    -1
     1
    -1
     1
     1
    -1
     1
     1
     1
     1
    -1
    -1
    -1
     1
    -1
     1
    -1
     1
    -1
    -1
     1
    -1
```

1
-1
1
-1
1
-1
1
-1
1
-1
1
-1
1
-1
-1
1
-1
1
-1
-1
1
1
-1
1
-1
1
-1
-1
1
-1
1
-1
1
-1
-1
1
-1
-1
-1
1
1
-1
1
-1
1
-1
1
1
1
1

```
-1
-1
-1
 1
-1
 1
-1
 1
 1
-1
 1
-1
-1
-1
 1
-1
-1
 1
-1
 1
-1
 1
 1
-1
 1
-1
 1
 1
-1
 1
-1
 1
-1
 1
 1
-1
-1
 1
 1
 1
-1
 1
 1
 1
 1
```

(c)

Which features seem to be most important? Justify your answer. Looking at the vector $\hat{w}$, it seems obvious that the first feature is the most important as it has the greatest weight. The second most important

feature seems to be the fourth feature, as it is the most negative, so it could be considered instrumental in classifying an expression as mad.

(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? I'd choose $w_1 = 0.9437$, $w_4 = -0.3922$ and $w_3 = 0.2664$. I'd zero out the other weights so that they'd have no contribution to our labels $\hat{y}$, and then simply do what we did for part (b) above, and if the value of $\hat{y}_i$ is > 0, we call it happy, and if it's < 0 we call it mad. If it's zero, we flip a coin.

(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 choose 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.

```
errors9 = zeros(8,1); % Prepare a vector of zeros

countErrors = @(x,y) sum(classify(x) ~= classify(y));           %
 Returns the number of errors in the set.
errorPercentage = @(x,y) (countErrors(x,y) / length(x));        %
 Returns the percentage error of the set.

% for set k = 1
k = 1;
testY1 = y(1:end - 16*k);
testX1 = X(1:end - 16*k, :);
holdoutY1 = y(end - 16*k + 1 : end);
holdoutX1 = X(end - 16*k + 1 : end, :);

w_hat1 = testX1\testY1;
y_hat1 = holdoutX1*w_hat1;

errors9(k) = errorPercentage(y_hat1, holdoutY1);

% for sets k = 2:7
for k = 2:7
    testYk = [y(1:end - 16*k) ; y(end - 16*(k - 1) + 1 : end)];
    testXk = [X(1:end - 16*k, :) ; X(end - 16*(k - 1) + 1 : end, :)];
    holdoutYk = y(end - 16*k + 1 : end - 16*(k - 1));
    holdoutXk = X(end - 16*k + 1 : end - 16*(k - 1), :);

    w_hatk = testXk\testYk;
    y_hatk = holdoutXk*w_hatk;

    errors9(k) = errorPercentage(y_hatk, holdoutYk);
end

% for set k = 8
```

```
k = k + 1
testY8 = y(end - 16*k + 1 : end);
testX8 = X(end - 16*k + 1 : end, :);
holdoutY8 = y(1 : end - 16*(k - 1));
holdoutX8 = X(1 : end - 16*(k - 1), :);

w_hat8 = testX8\testY8;
y_hat8 = holdoutX8*w_hat8;

errors9(k) = errorPercentage(y_hat8, holdoutY8)

mean(errors9)
```

```
k =

     8


errors9 =

         0
         0
    0.0625
         0
    0.0625
    0.1250
    0.0625
         0


ans =

    0.0391
```

(f)

What is the estimated error rate using all 9 features? What is it using the 3 features you chose in (d) above? The mean error with 9 features is 0.0391. The mean error with 3 features is 0.0859. This seems right; more data helps the prediction to correctly place data it hasn't seen before.

```
errors3 = zeros(8,1); % Prepare a vector of zeros

% for set k = 1
k = 1;
testY1 = y(1:end - 16*k);
testX1 = X(1:end - 16*k, :);
holdoutY1 = y(end - 16*k + 1 : end);
holdoutX1 = X(end - 16*k + 1 : end, :);

w_hat1 = testX1\testY1;
w_hat1 = [w_hat1(1) 0 w_hat1(3) w_hat1(4) 0 0 0 0 0]';
y_hat1 = holdoutX1*w_hat1;
```

```
errors3(k) = errorPercentage(y_hat1, holdoutY1);

% for sets k = 2:7
for k = 2:7
    testYk = [y(1:end - 16*k) ; y(end - 16*(k - 1) + 1 : end)];
    testXk = [X(1:end - 16*k, :) ; X(end - 16*(k - 1) + 1 : end, :)];
    holdoutYk = y(end - 16*k + 1 : end - 16*(k - 1));
    holdoutXk = X(end - 16*k + 1 : end - 16*(k - 1), :);

    w_hatk = testXk\testYk;
    w_hatk = [w_hatk(1) 0 w_hatk(3) w_hatk(4) 0 0 0 0 0]';
    y_hatk = holdoutXk*w_hatk;

    errors3(k) = errorPercentage(y_hatk, holdoutYk);
end

% for set k = 8
k = k + 1
testY8 = y(end - 16*k + 1 : end);
testX8 = X(end - 16*k + 1 : end, :);
holdoutY8 = y(1 : end - 16*(k - 1));
holdoutX8 = X(1 : end - 16*(k - 1), :);

w_hat8 = testX8\testY8;
w_hat8 = [w_hat8(1) 0 w_hat8(3) w_hat8(4) 0 0 0 0 0]';
y_hat8 = holdoutX8*w_hat8;

errors3(k) = errorPercentage(y_hat8, holdoutY8)

mean(errors3)


k =

     8


errors3 =

         0
         0
         0
         0
         0
    0.1875
    0.0625
    0.4375


ans =

    0.0859
```

Published with MATLAB® R2016b