

1)

A)

1)

a)

$$X = \begin{bmatrix} 2.5 & 2.4 \\ 0.5 & 0.7 \\ 2.2 & 2.9 \\ 1.9 & 2.2 \\ 3.1 & 3.0 \\ 2.3 & 2.7 \\ 2.0 & 1.6 \\ 1.0 & 1.1 \\ 1.5 & 1.6 \\ 1.1 & 0.9 \end{bmatrix}$$

$$\text{mean} = \text{mean}(X) = \begin{bmatrix} 1.81 & 1.91 \end{bmatrix}$$

$$\text{cov} = \frac{1}{n} \begin{bmatrix} \sum (X_1 - m(1))(X_1 - m(1))^T & \sum (X_1 - m(1))(X_2 - m(2))^T \\ \sum (X_2 - m(2))(X_1 - m(1))^T & \sum (X_2 - m(2))(X_2 - m(2))^T \end{bmatrix} = \begin{bmatrix} 0.5549 & 0.5539 \\ 0.5539 & 0.6449 \end{bmatrix}$$

$$\begin{vmatrix} 0.5549 - \lambda & 0.5539 \\ 0.5539 & 0.6449 - \lambda \end{vmatrix} = 0$$

Solving

$$\lambda_1 = 1.1556$$

$$\lambda_2 = 0.0442$$

$$\begin{bmatrix} 0.5549 & 0.5539 \\ 0.5539 & 0.6449 \end{bmatrix} \begin{bmatrix} V_{11} \\ V_{12} \end{bmatrix} = \begin{bmatrix} 1.1556 V_{11} \\ 1.1556 \\ \cancel{0.0442} V_{22} \end{bmatrix}$$

$$\begin{aligned} 0.5549(V_{11}) + 0.5539(V_{12}) - 1.1556(V_{11}) &= 0 \\ 0.5539(V_{11}) + 0.6449(V_{12}) - \cancel{0.0442}(V_{22}) &= 0 \end{aligned}$$

$$\begin{aligned} -0.6007 V_{11} + 0.5539 V_{12} &= 0 \\ 0.5539 V_{11} &= \cancel{0.6007} V_{12} \end{aligned}$$

$$\therefore V_{11} = \frac{-0.5539 V_{12}}{-0.6007}$$

(or)

$$V_{11} = \frac{0.5107 V_{12}}{0.5539}$$

$$\Rightarrow \boxed{V_{11} = -0.9221 V_{12}}$$

$$\text{let } \begin{aligned} V_{12} &= 0.7352 \\ V_{11} &= 0.6779 \end{aligned} \quad \} \text{ (using MATLAB)}$$

$$\begin{bmatrix} 0.5549 & 0.5539 \\ 0.5539 & 0.6449 \end{bmatrix} \begin{bmatrix} V_{21} \\ V_{22} \end{bmatrix} = \begin{bmatrix} 0.442 V_{21} \\ 0.442 V_{22} \end{bmatrix}$$

$$\begin{aligned} 0.5549 V_{21} + 0.5539 V_{22} - 0.442 V_{21} &= 0 \\ 0.5539 V_{21} + 0.6449 V_{22} - 0.442 V_{22} &= 0 \end{aligned}$$

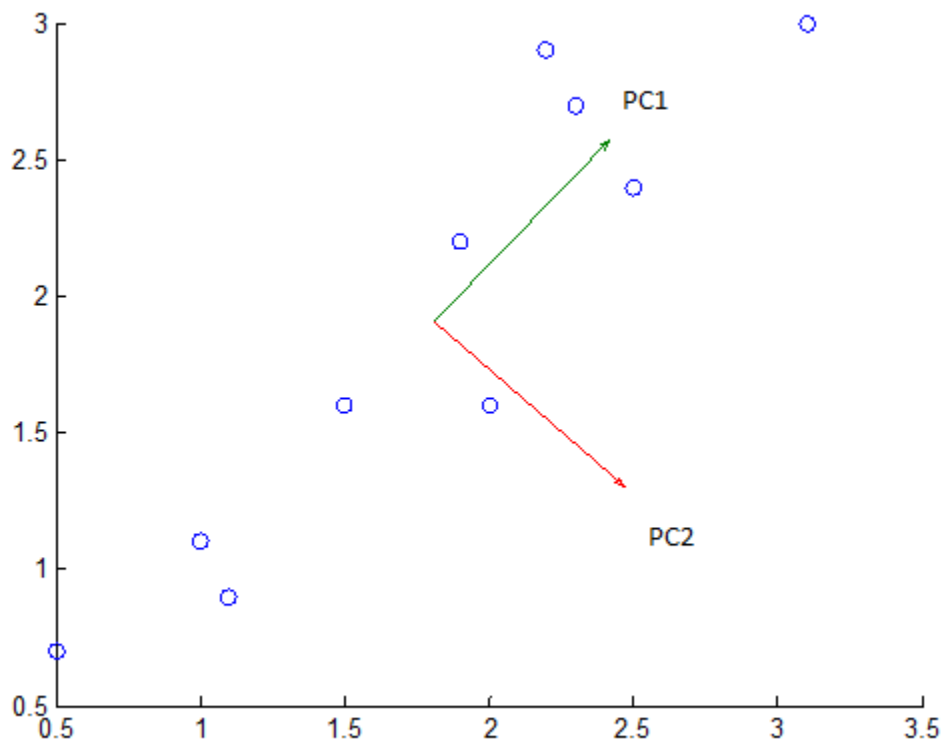
$$\therefore V_{21} = 1.0846 V_{22}$$

$$\Rightarrow \text{let } v_{21} = 0.7352$$

$$\text{then } v_{22} = -0.6779$$

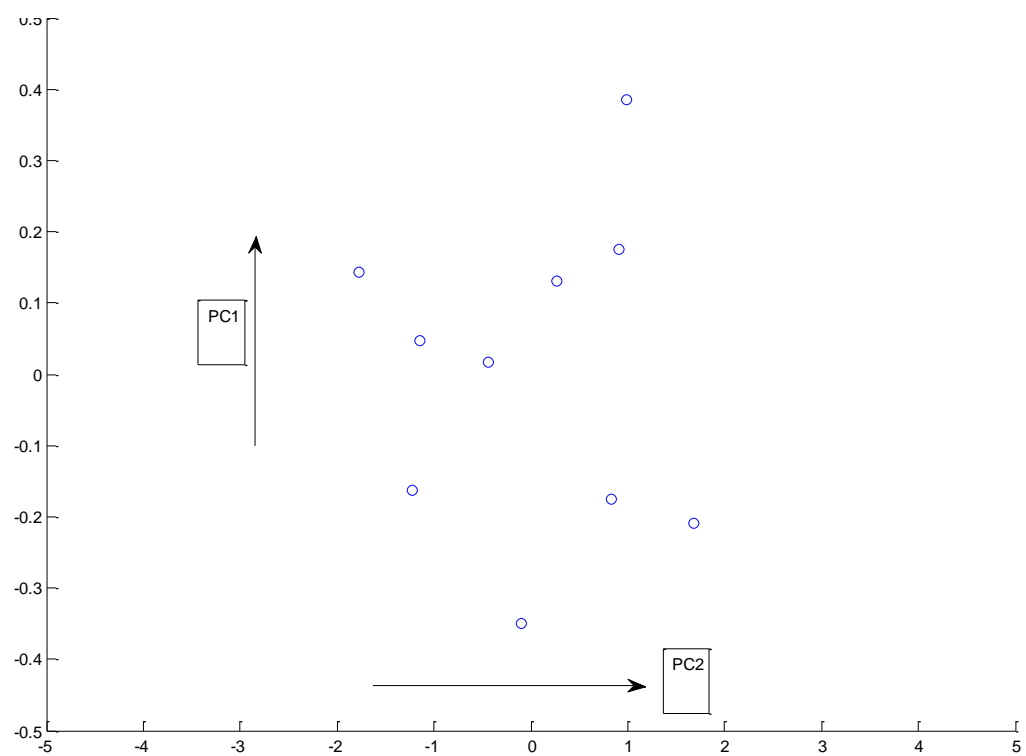
$$\therefore e_1 = \begin{bmatrix} 0.6779 \\ 0.7352 \end{bmatrix}$$

$$e_2 = \begin{bmatrix} 0.7352 \\ -0.6779 \end{bmatrix}$$



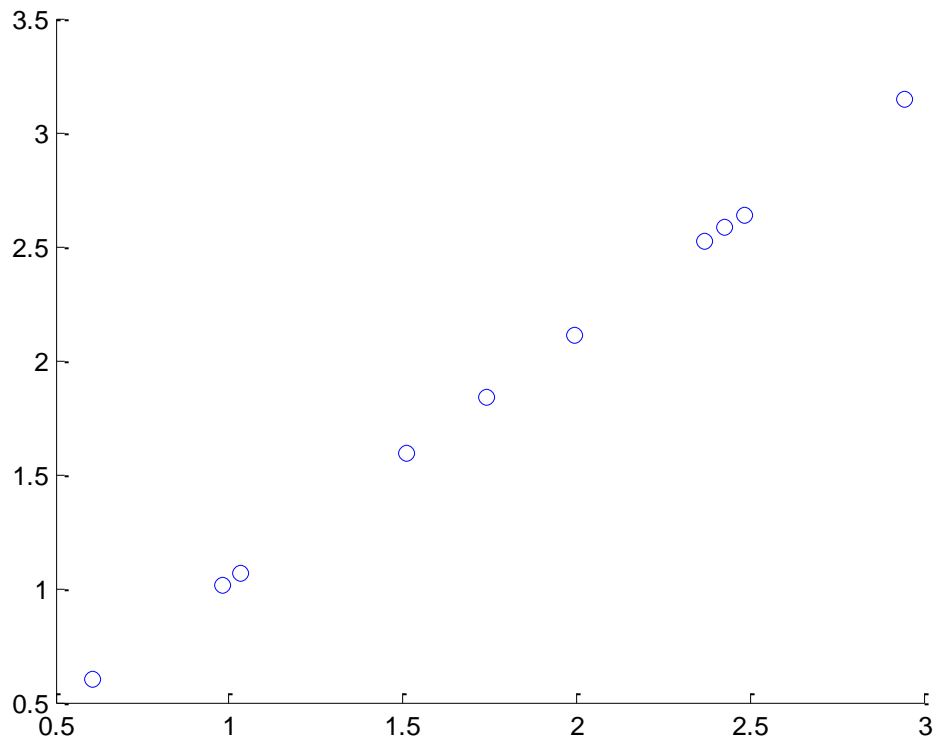
b)

a1	a2
0.82797	0.175115
-1.77758	-0.14286
0.992197	-0.38437
0.27421	-0.13042
1.675801	0.209498
0.912949	-0.17528
-0.09911	0.349825
-1.14457	-0.04642
-0.43805	-0.01776
-1.22382	0.162675



c) New representation:

2.37125896400000	2.51870600832217
0.605025583745627	0.603160886338143
2.48258428755000	2.63944241997847
1.99587994658902	2.11159364495307
2.94598120291464	3.14201343391850
2.42886391124136	2.58118069424077
1.74281634877673	1.83713685698813
1.03412497746524	1.06853497544495
1.51306017656077	1.58795783010856
0.980404601156605	1.01027324970724



Range/distance= 3.4534

2)

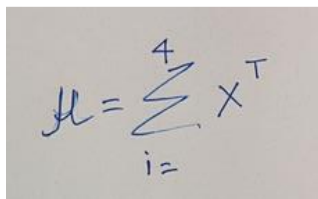
a)

Let X=

[-2 1 2 -3 4 1 0 3 0 2 1 1 2 3 -2 -3 2 1 0
1 2 -4 2 -4 2 5 2 2 1 -3 0 0 1 -2 1 1 -3 -2
1 -3 2 1 0 -3 -5 -1 3 3 -2 -3 -2 -1 1 0 5 4 2
3 -1 0 2 2 -5 -4 -1 2 -1 3 4 4 2 1 2 -2 1 -1]

Rearranging the matrix (ie X^T)

-2	1	1	3
1	2	-3	-1
2	-4	2	0
-3	2	1	2
4	-4	0	2
1	2	-3	-5
0	5	-5	-4
3	2	-1	-1
0	2	3	2
2	1	3	-1
1	-3	-2	3
1	0	-3	4
2	0	-2	4
3	1	-1	2
-2	-2	1	1
-3	1	0	2
2	1	5	-2
1	-3	4	1
0	-2	2	-1


$$\mu = \sum_{i=1}^4 X^T$$

Mean= [0.75 -0.25 0 0.5 0.5 -1.25 -1 0.75 1.75 1.25 -0.25 0.5 1 1.25 -0.5 0 1.5 0.75 -0.25]

To Find the A matrix we subtract mean to each of the X values

$$A = [x - \mu]$$

-2.8	1.25	2	-3.5	3.5	2.25	1	2.25	-1.8	0.75	1.25	0.5	1	1.75	-1.5	-3	0.5	0.25	0.25
0.25	2.25	-4	1.5	-4.5	3.25	6	1.25	0.25	-0.3	-2.8	-0.5	-1	-0.3	-1.5	1	-0.5	-3.8	-1.8
0.25	-2.8	2	0.5	-0.5	-1.8	-4	-1.8	1.25	1.75	-1.8	-3.5	-3	-2.3	1.5	0	3.5	3.25	2.25
2.25	-0.8	0	1.5	1.5	-3.8	-3	-1.8	0.25	-2.3	3.25	3.5	3	0.75	1.5	2	-3.5	0.25	-0.8

Hence $C = A^T \times A$

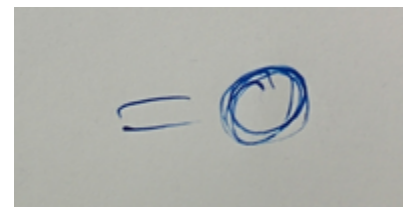
$$C = \sum_{i=1}^n [x - \mu]^T [x - \mu]$$

Hence the **SORTED** 4X4 inner product matrix in **descending order based on Eigen value** is

[69.87500000000000	-18.87500000000000	-26.37500000000000	-24.62500000000000
-18.87500000000000	121.37500000000000	-53.12500000000000	-49.37500000000000
-26.37500000000000	-53.12500000000000	98.37500000000000	-18.87500000000000
-24.62500000000000	-49.37500000000000	-18.87500000000000	92.87500000000000]

After sorting the minimum squared error representations of these samples in 3D space is nothing but the eigen values by multiplying the new eigenvectors with the x data-the mean.

69.875000- λ	-18.875000	-26.37500	-24.6250
-18.875000	121.37500- λ	-53.12500	-49.3750
-26.375000	-53.125000	98.37500- λ	-18.8750
-24.625000	-49.37500	-18.8750	92.8750- λ



Solving Eigen vectors and Eigen values:

Eigen Vectors

[0.5000000000000000	0.862499590592227	-0.0412458686483948	-0.0662814796732877
0.5000000000000000	-0.347332077693905	0.0682250237569957	-0.790383308236069
0.5000000000000000	-0.220461654810414	0.691237386257788	0.472850435759295
0.5000000000000000	-0.294705858087909	-0.718216541366389	0.383814352150061]

Eigen Values

0.0000

92.6318

114.3171

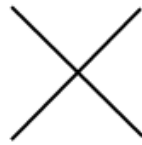
175.5512

Now= $A \cdot \text{sorted_evec}$; and normalizing the vectors gives us the answer

U1	U2	U3
0.072944	-0.12277	-0.33008
-0.26034	-0.11787	0.116777
0.299985	0.096062	0.27777
-0.01068	-0.04536	-0.42517
0.27654	-0.1753	0.441571
-0.37621	0.150822	0.239258
-0.59258	-0.02265	0.056571
-0.19897	0.003712	0.250194
0.045693	0.072366	-0.20214
0.008437	0.259791	0.105043
0.189486	-0.35382	0.151831
0.003806	-0.4665	0.035852
0.034491	-0.40571	0.102561
-0.05241	-0.20419	0.194421
0.193968	-0.00757	-0.16058
0.01329	-0.11639	-0.36617
0.050845	0.456266	0.089851
0.345678	0.168427	0.075634
0.161715	0.183713	0.056984

b) Multiplying the following vectors ie($U^T * X2 - T$) we get the vector of weights needed

0.072944	-0.12277	-0.33008
-0.26034	-0.11787	0.116777
0.299985	0.096062	0.27777
-0.01068	-0.04536	-0.42517
0.27654	-0.1753	0.441571
-0.37621	0.150822	0.239258
-0.59258	-0.02265	0.056571
-0.19897	0.003712	0.250194
0.045693	0.072366	-0.20214
0.008437	0.259791	0.105043
0.189486	-0.35382	0.151831
0.003806	-0.4665	0.035852
0.034491	-0.40571	0.102561
-0.05241	-0.20419	0.194421
0.193968	-0.00757	-0.16058
0.01329	-0.11639	-0.36617
0.050845	0.456266	0.089851
0.345678	0.168427	0.075634
0.161715	0.183713	0.056984



1	0.75
2	-0.25
-4	0
2	0.5
-4	0.5
2	-1.25
5	-1
2	0.75
2	1.75
1	1.25
-3	-0.25
0	0.5
0	1
1	1.25
-2	-0.5
1	0
1	1.5
-3	0.75
-2	-0.25

Hence the weights needed are

$$X2 = \text{mean} + 10.4722412661684(e1) + 0.729456174057581(e2) + -3.34291138878802(e3)$$

c) Mean square error is calculated by reconstructing the data similar to as shown above, finding the difference between the reconstructed data and the new data, squaring it and finding the square root of the sum of the errors. Finally we can divide the errors by the number of dimensions

$$\omega 1 = u1^T * (X1 - T)$$

$$\omega 2 = u1^T * (X2 - T)$$

$$\omega 3 = u1^T * (X3 - T)$$

$$\omega 4 = u1^T * (X4 - T)$$

$\text{newx1} = T + u1 * w1;$
 $\text{newx2} = T + u1 * w2;$
 $\text{newx3} = T + u1 * w3;$
 $\text{newx4} = T + u1 * w4;$

Therefore the newx

-2	1	1	3
1	2	-3	-1
2	-4	2	-1.44E-15
-3	2	1	2
4	-4	1.78E-15	2
1	2	-3	-5
-1.55E-15	5	-5	-4
3	2	-1	-1
2.22E-16	2	3	2
2	1	3	-1
1	-3	-2	3
1	2.89E-15	-3	4
2	2.22E-15	-2	4
3	1	-1	2
-2	-2	1	1
-3	1	-1.22E-15	2
2	1	5	-2
1	-3	4	1
5.55E-16	-2	2	-1

Hence the mean square errors for each sample are:

4.112e-31, 2.732e-30, 9.0952e-31, 3.8580e-30

d) Using only 2-D

3.6268 0.5881 0.2369 0.4234

e) The Euclidian distance to each projected image are:

12.0711 3.2728 16.1466 15.0901

The answer is the 2nd sample

f) The new distances are

16.6073 4.1098 18.1979 15.3917

The answer is still the 2nd sample

This does make intuitive sense because the Euclidian distance between the test vector and the 2nd sample increases. This is because using all the Eigen values gives a more accurate prediction of the mean square errors and the distance from the original values.

3)

```
%% Find the eigenvalues and eigen vectors
T=mean(X,2);
A=bsxfun(@minus,X,T);
C=A'*A;
[vec,eval]=eig(C);
d1 = diag(eval);
v1=vec;

%% sort the eigen vectors and values
[sorted_eval, e_index] = sort(d1,'descend');
sorted_vec = zeros(165);
for i=1:165
    sorted_vec(:,i) = v1(:,e_index(i));
end

%% U is the eigenfaces
u=A*sorted_vec;

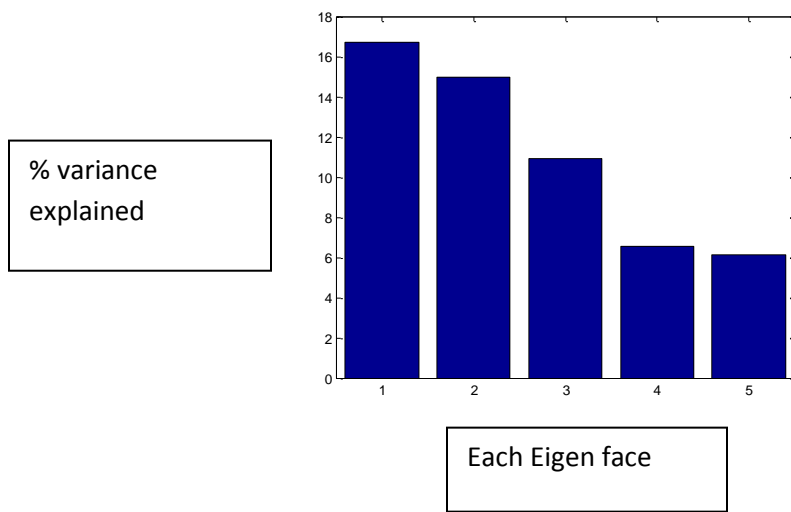
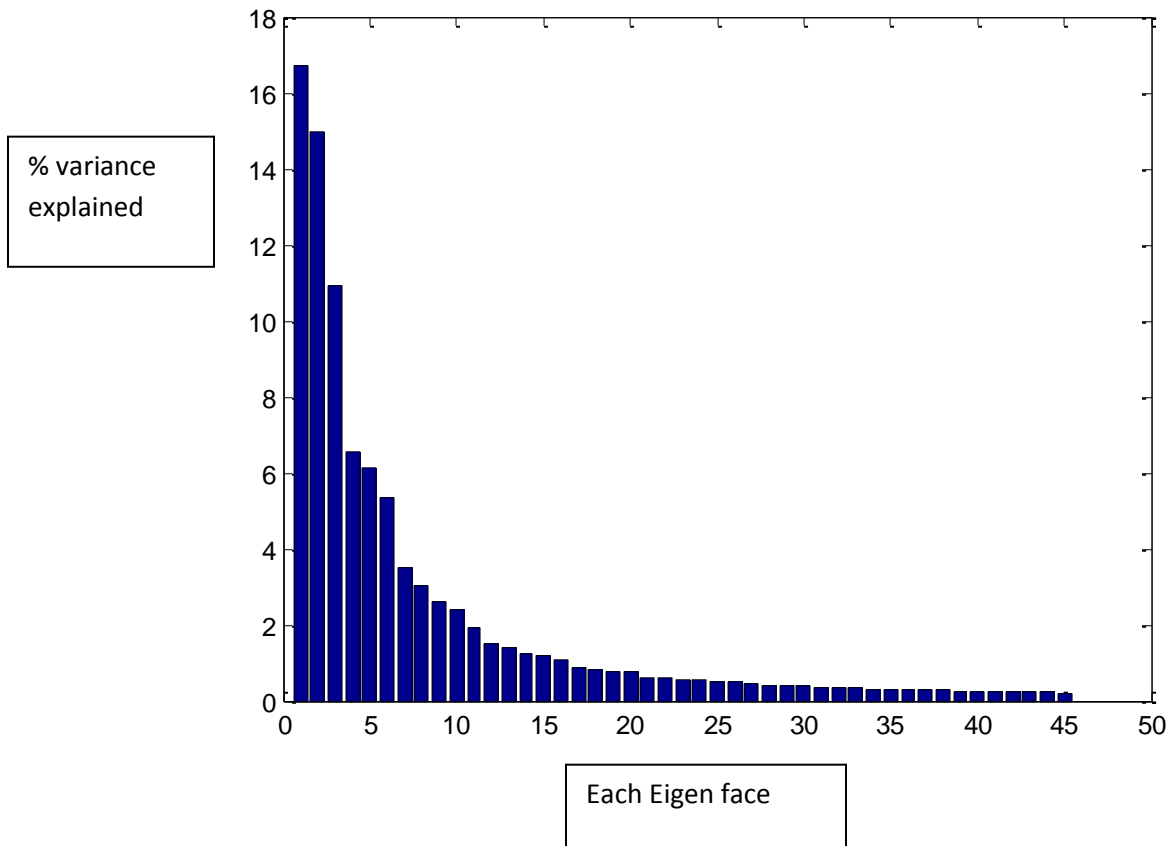
%% percent has the total percentages given by the eigenvalues
for i=1:length(sorted_eval)
    percent(i)=(sum(sorted_eval(1:i,1))/sum(sorted_eval))*100;
end

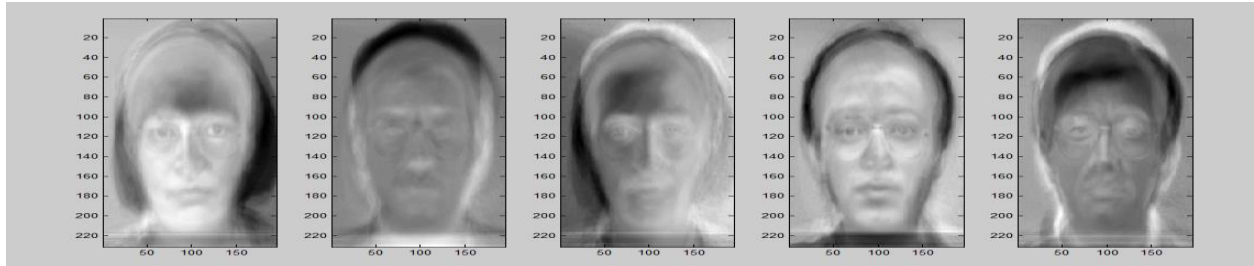
%% n_eigenfaces has the no.of eigenfaces needed for 90%
n_eigenfaces=min(find(percent(1,:)>90));

%% percent_each has the each percentages given by the eigenvalues
for i=1:length(sorted_eval)
    percent_each(i)=(sorted_eval(i)/sum(sorted_eval))*100;
end
bar(1:n_eigenfaces+10,percent_each(1,1:n_eigenfaces+10))
figure
bar(1:5,percent_each(1,1:5))
```

3)

a) The new number of dimension= 35





b) original image



c) accuracy=80%

Testing values	Actual values	R/W
1	1	TRUE
4	1	FALSE
2	2	TRUE
2	2	TRUE
3	3	TRUE
7	3	FALSE
4	4	TRUE
4	4	TRUE
5	5	TRUE
5	5	TRUE
1	6	FALSE
6	6	TRUE
7	7	TRUE
7	7	TRUE
2	8	FALSE
8	8	TRUE
9	9	TRUE
9	9	TRUE
10	10	TRUE
10	10	TRUE
11	11	TRUE
11	11	TRUE
12	12	TRUE
5	12	FALSE
13	13	TRUE
13	13	TRUE
4	14	FALSE
14	14	TRUE
15	15	TRUE
15	15	TRUE

d) Accuracy increased to 83.33%

Testing values	Actual values	R/W
1	1	1
4	1	0
2	2	1
2	2	1
3	3	1
15	3	0
4	4	1
4	4	1
5	5	1
5	5	1
6	6	1
6	6	1
7	7	1
7	7	1
2	8	0
8	8	1
9	9	1
9	9	1
10	10	1
10	10	1
11	11	1
11	11	1
12	12	1
5	12	0
13	13	1
13	13	1
4	14	0
14	14	1
15	15	1
15	15	1

Yes this is what I would expect as there is no information that is lost in the original dimension space. Hence the algorithm should perform better.