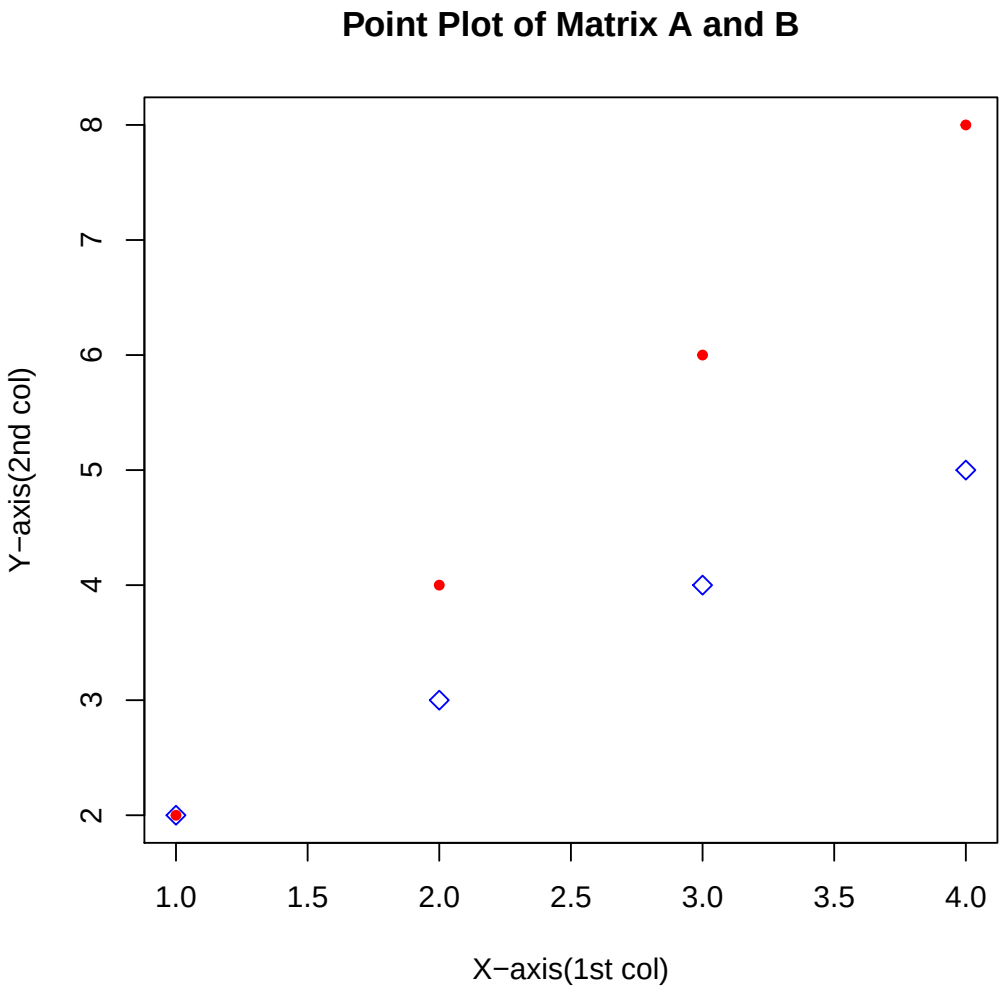
AMS 550.400 Homework 3 Due Date: Nov 19

Last Compiled on November 28, 2012

Solution for Ex 7.18

(a) Use Sweave to accomplish goal, we can get the figure: jhuang63HW3-1.png (uploaded to github file)



Listing 1: R Codes for Ex 7.18

```
\includegraphics{jhuang63HW3-002}
```

(b) The rank of A is 2 and the rank of B is 1.
Generating the second and third columns of A from the first column will change the determinant of this matrix. So, it does not work.

(c) See the following description:

```
> a = cbind(A[1:2, 3])
> A12 = A[1:2, 1:2]
> dA12 = det(A12)
> x = solve(A12, a)
```

Here is how the linear combination could create the third column from the first two:
For A:

$$c3 = -1 * c1 + 2 * c2.$$

For B:

$$c3 = x * c1 + y * c2$$

for all x and y that satisfy $1 * x + 2 * y = 3$.

The determinant of the rank 2 submatrix $A12$ is $\det(A12) = \det \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} = -1 \neq 0$,

So, this matrix can be used with corresponding coordinates in the third column to derive the linear combination.

For B, the rank is 1, so we can use the first row of it to find the linear combination. The coefficient for the 1st column x and the 2nd column y can be any numbers that satisfy following demand:

$$1 * x + 2 * y = 3$$

Ex 7.24

- `[COMMIT]` use `lstlisting` to list your R code
- `[COMMIT]` use R/Sweave for computation, but do not use the built-in `dist` function

---

- `[COMMIT]` use R/Sweave for computation, and this time, do use the built-in `dist` function for comparison

---

- `[COMMIT]` make sure to explain your computation, e.g., compare the two computations

Solution for Ex 7.24

Listing 2: R codes for Ex 7.24

```
M=A%*%t(A)
c=cbind(diag(M))
one=cbind(c(1,1,1,1))
D=c%*%t(one)+one%*%t(c)-2*M
Dis=sqrt(D)
```

- Without Built-in dist function:

```
> M = A %*% t(A)
> c = cbind(diag(M))
> one = cbind(c(1, 1, 1, 1))
> D = c %*% t(one) + one %*% t(c) - 2 * M
> Dis = sqrt(D)
```

- With Built-in dist function:

```
> di = dist(M)
> distance = as.matrix
```

Now, we can calculate the distance matrix:

$$D = \begin{pmatrix} 0 & 3 & 12 & 27 \\ 3 & 0 & 3 & 12 \\ 12 & 3 & 0 & 3 \\ 27 & 12 & 3 & 0 \end{pmatrix}$$

If there are no built-in dist functions can be used, then we shall use the formula in the book to derive a matrix $M = AA'$,then calculate the distance matrix from the diagonal entries of X, matrix X as well as a one vector.

If there are built-in dist funtions can be used, we just need to use the built-in R DIST function to do the calculation directly.

Ex 7.30 Omit (c), (d) and (e). The necessary data is saved in `matlabclown.RData` and can be found from the course git folder. The followings are the equivalent R version:

```
load('matlabclown.RData')
image(X) # omit this in your Sweave code
svdX = svd(X)
```

```
U = svdX$u
S = diag(svdX$d)
V = svdX$v
k = 10
M = U[,1:k,drop=FALSE] %*% S[1:k,1:k,drop=FALSE] %*% t(V[,1:k,drop=FALSE])
image(M) # omit this in your Sweave code
image(M,col=gray.colors(k))
```

(a) [COMMIT] choose a small, a medium and a large value for k

- for each k,
  - do [COMMIT]
  - your performance evaluation is to be included as a caption, and change `tinyK`, `width` and `height` accordingly

```
\begin{figure}
\centering
\begin{Schunk}
\begin{Sinput}
> tinyK = 1
\end{Sinput}
\end{Schunk}
\includegraphics{jhuang63HW3-009}
\caption{<YOUR PERFORMANCE EVALUATION> on $1$}
\label{fig:matlabclownKaNumber}
\end{figure}
```

(b)
- [COMMIT] code up all your computation using R/Sweave before starting to type your explanation
- [COMMIT] write your explanation referring to the numbers computed in the previous step, using

Solution for Ex 7.30:

(b)

(c) Based on the four figures we have now, it is obvious that clown pictures become clearer as the value of K increasing from small to big. And the corresponding rate are 0.204574122478363,0.574105688285 respectively.

Problems from Chapter 4: Multidimensional Scaling

(a)
```
> tinyK = 1
> smallK = 20
> mediumK = 80
> largeK = 150
> load("/Users/hj//550400/jhuang63HW3.git/matlabclown.RData")
> svdX = svd(X)
> U = svdX$u
> S = diag(svdX$d)
> V = svdX$v
> k = tinyK
> M = U[, 1:k, drop = FALSE] %*% S[1:k, 1:k, drop = FALSE] %*%
+     t(V[, 1:k, drop = FALSE])
> image(M, col = gray.colors(k))
```

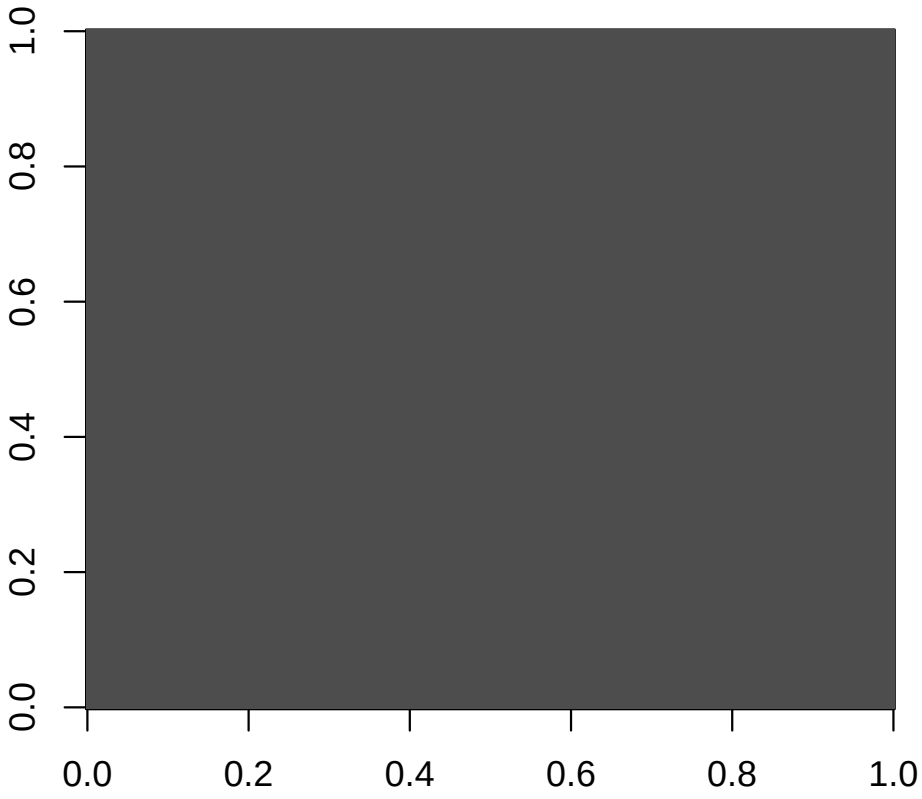


Figure 1: The Poorest Performance on 1

```
> tinyK = 1
> smallK = 20
> mediumK = 80
> largeK = 150
> load("/Users/hj//550400/jhuang63HW3.git/matlabclown.RData")
> svdX = svd(X)
> U = svdX$u
> S = diag(svdX$d)
> V = svdX$v
> k = smallK
> M = U[, 1:k, drop = FALSE] %*% S[1:k, 1:k, drop = FALSE] %*%
+     t(V[, 1:k, drop = FALSE])
> image(M, col = gray.colors(k))
```

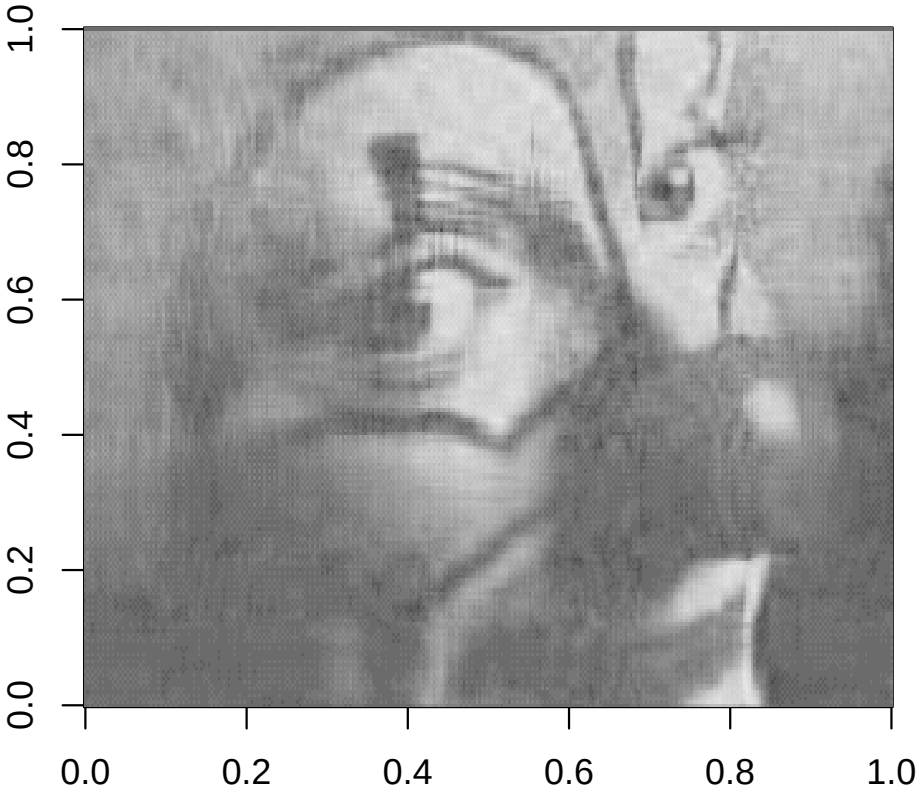


Figure 2: Poor Performance on 20

```
> tinyK = 1
> smallK = 20
> mediumK = 80
> largeK = 150
> load("/Users/hj//550400/jhuang63HW3.git/matlabclown.RData")
> svdX = svd(X)
> U = svdX$u
> S = diag(svdX$d)
> V = svdX$v
> k = mediumK
> M = U[, 1:k, drop = FALSE] %*% S[1:k, 1:k, drop = FALSE] %*%
+     t(V[, 1:k, drop = FALSE])
> image(M, col = gray.colors(k))
```

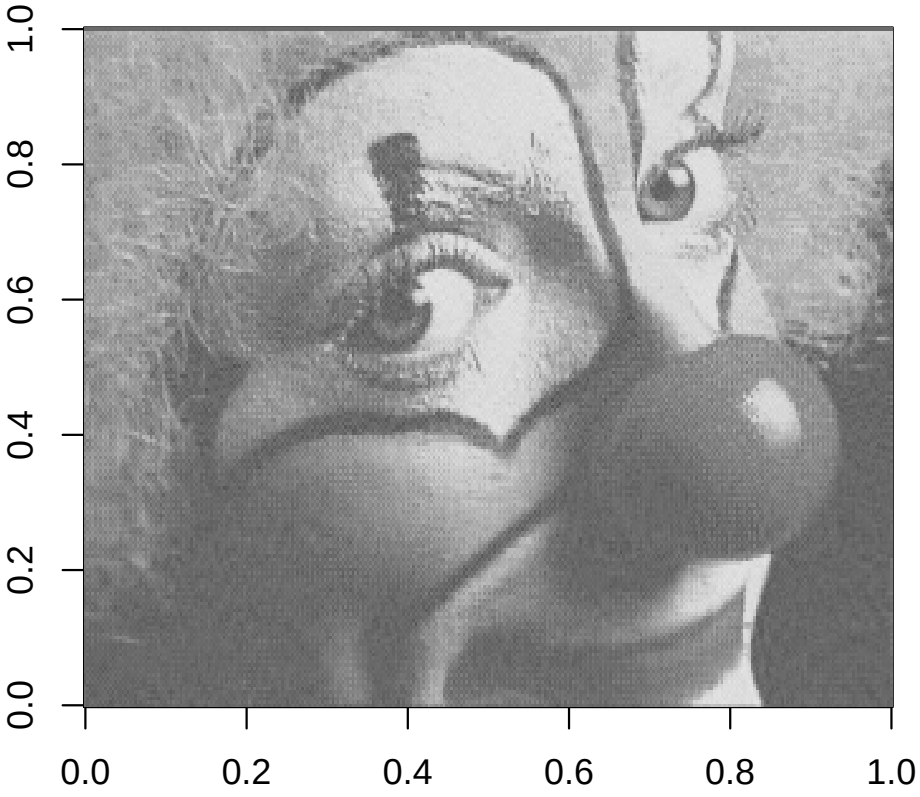


Figure 3: Good Performance on 80

```
> tinyK = 1
> smallK = 20
> mediumK = 80
> largeK = 150
> load("/Users/hj//550400/jhuang63HW3.git/matlabclown.RData")
> svdX = svd(X)
> U = svdX$u
> S = diag(svdX$d)
> V = svdX$v
> k = largeK
> M = U[, 1:k, drop = FALSE] %*% S[1:k, 1:k, drop = FALSE] %*%
+     t(V[, 1:k, drop = FALSE])
> image(M, col = gray.colors(k))
```

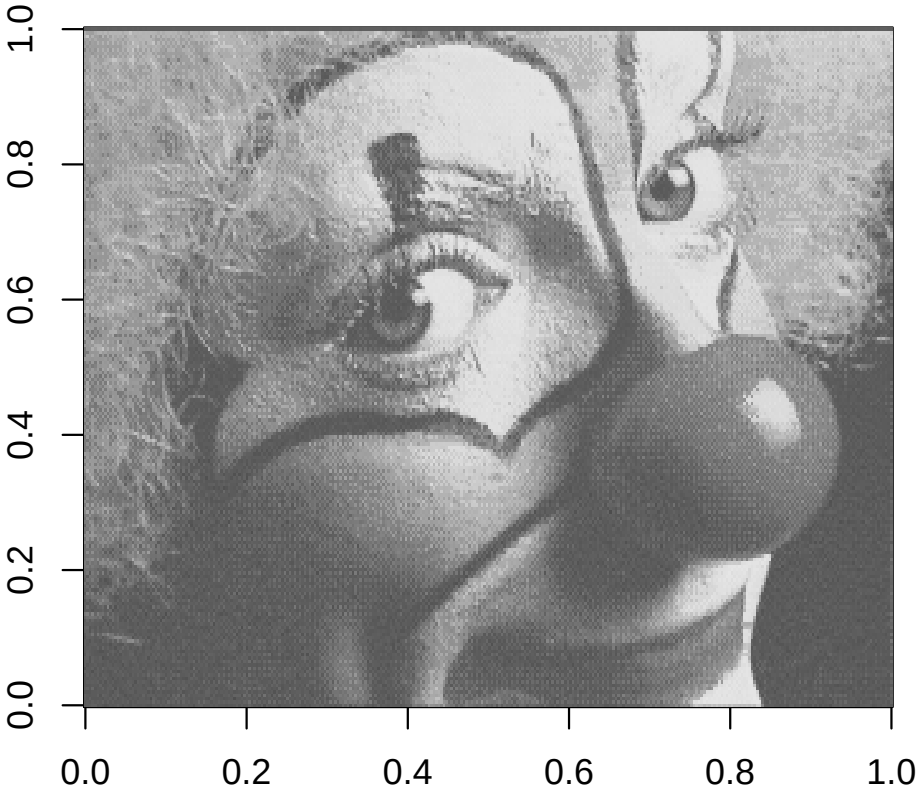


Figure 4: Better Performance on 150

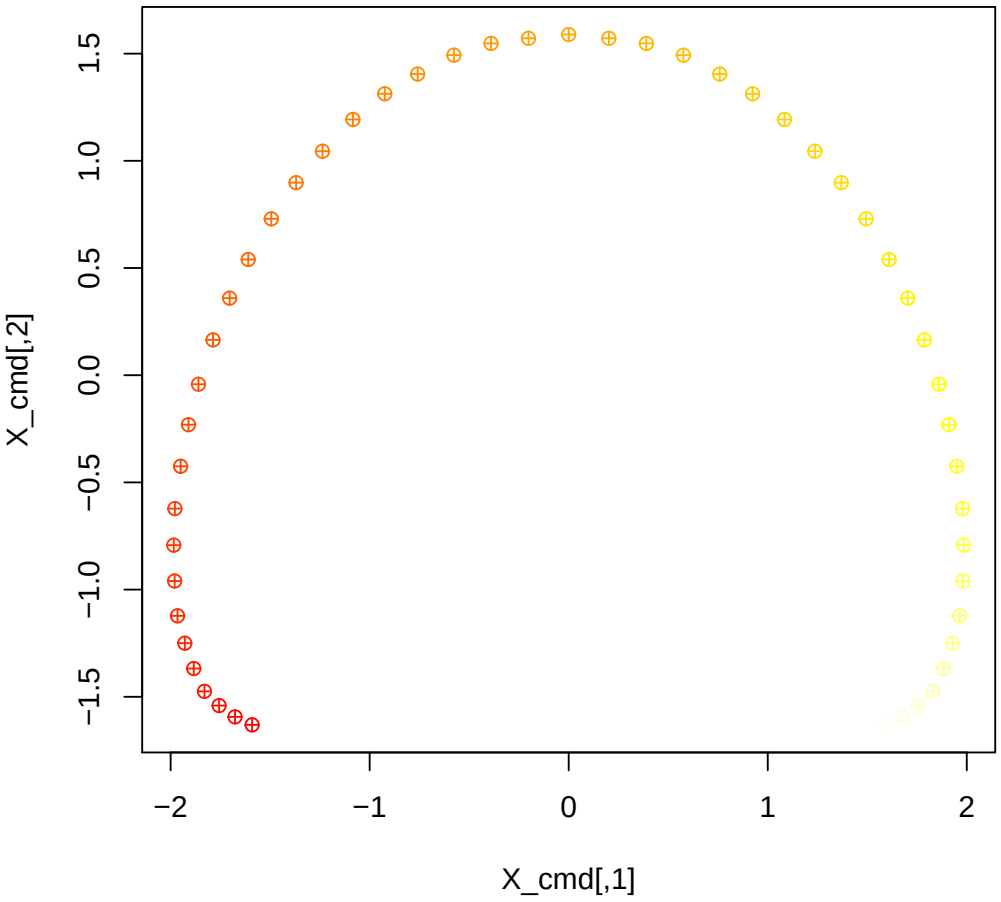
Solution for Ex 4.1:

$(1,1)$	$(2,1)$	$(3,1)$	$(4,1)$	$(5,1)$	$(6,1)$	$(7,1)$	$(8,1)$	$(9,1)$	$(10,1)$

Figure 5: first ten objects on a Line

Listing 3: R codes

```
M=matrix(rep(1,51*51),nrow=51,byrow=TRUE)
for (i in 1:51){
  for (j in 1:51){
    if(i == j) M[i,j]=9
    else if(abs(i-j)>=22 & abs(i-j)<=24) M[i,j]=1
    else if(abs(i-j)>=19 & abs(i-j)<=21) M[i,j]=2
    else if(abs(i-j)>=16 & abs(i-j)<=18) M[i,j]=3
    else if(abs(i-j)>=13 & abs(i-j)<=15) M[i,j]=4
    else if(abs(i-j)>=10 & abs(i-j)<=12) M[i,j]=5
    else if(abs(i-j)>=7 & abs(i-j)<=9) M[i,j]=6
    else if(abs(i-j)>=4 & abs(i-j)<=6) M[i,j]=7
    else if(abs(i-j)>=1 & abs(i-j)<=3) M[i,j]=8
    else M[i,j]=0
  }
}
D=matrix(rep(1,51*51),nrow=51,byrow=TRUE)
for (i in 1:51){
  for (j in 1:51){
  D[i,j]=sqrt(M[i,i]+M[j,j]-2*M[i,j])
  }
}
X_cmd=cmdscale(D)
plot(X_cmd,col=heat.colors(51),pch=10)
```



From the graphic above, every point is similar to its neighbors while the similarity dies down with distance. The disimilarity in colar represents the disimilarity of each point.

Ex 4.3

- `[COMMIT]` list your R code using `lstlisting`
- `[COMMIT]` load the data (`require(MVA);data(gardenflowers)`) and compute using R/Sweave
- `[COMMIT]` include a plot of (relative) positions using R/Sweave
- `[COMMIT]` allocate at least a quarter page of *text* explaining the result

Solution for Ex 4.3:

Listing 4: R codes for Flowers Comparison

```
require(MVA)
data(gardenflowers)
flowers_m=cmdscale(gardenflowers,k=17,eig=TRUE)
x=flowers_m$points[,1]
```

```
y=flowers_m$points[,2]
plot(x,y,xlab = "First column of input", ylab = "Second column of input",xlim = range(x)*1.5, type = "n")
text(x, y, labels = colnames(as.matrix(gardenflowers)), cex = 0.5)
```