Assignment 2

Tiffany Woodley

October 2018

1 Question 1

1.1

When forcing an intercept this limits the fit of the line, as there is now only one parameter to tune in order to try and find the line of best fit. Unless the chosen intercept matches the target function's intercept, the fitted line would be less likely to approximate the target function. With an unknown target function the possibility of knowing the intercept's exact value would be unlikely.

However, in this case with the target function being known, the value of the intercept equals 0. Both of the fitted lines with set intercepts are close to the target line, but which line would generalize better would depend on the noise in the given case

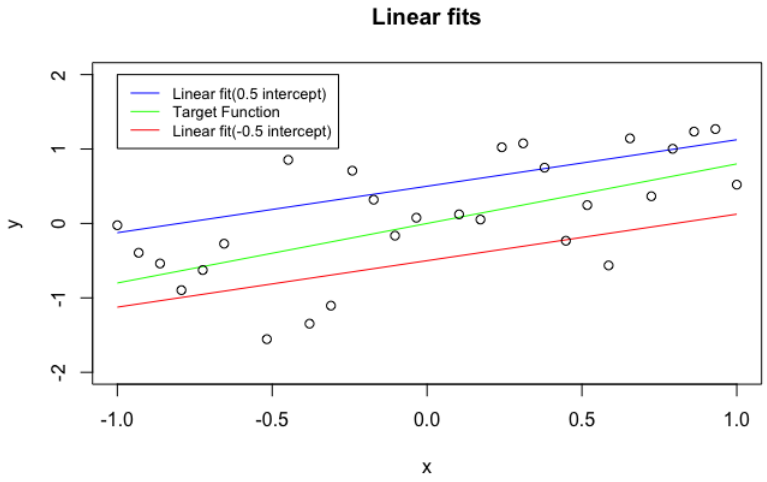

Figure 1: Linear Fits

1.2

The validation error typically underestimates the true generalization error of the fit. The more training data that is used to train the model, the better the chance that the model has to estimate the target function and reduce the generalization error. This can be seen in the graph below, as the validation set increases in turn decreasing the size of the training set, the error grows at a increasing rate. The validation error (red line) starts to approximate the expected generalization error (blue line) when there are more observations, this is due to the more observations helping the model better approximate the target function.

This is why a method such as k-folds validation is a better way of approximating the generalization error. As you are training and testing on multiple sets. This allows for a smaller validation set size and averages the error over multiple iterations.

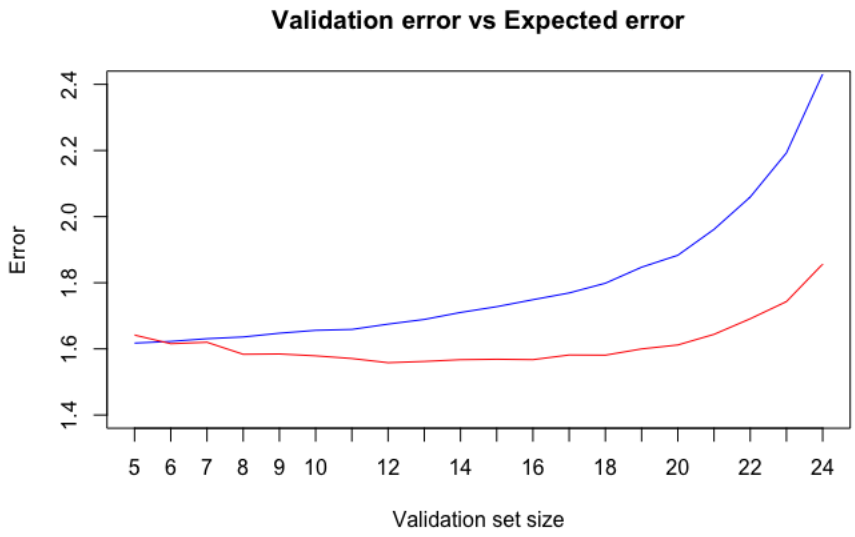

Figure 2: Validation error vs Expected error

2 Question 2

2.1

The graph below shows the target function(sine wave) we are going to try approximate using a Legendre polynomial function in red. The dots represent the samples of this function with added noise.

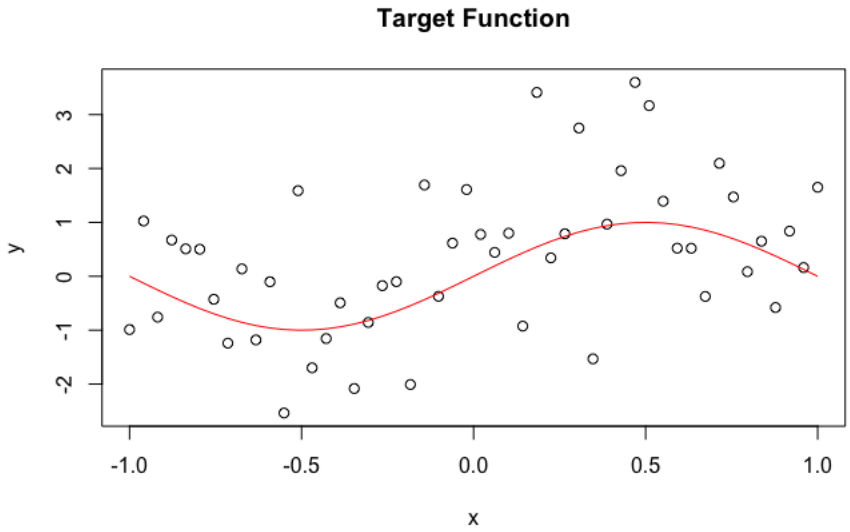

Figure 3: Sine wave target function

2.2

The graph below shows how regularization affects the fit of a model. Fitting a ten degree Legendre polynomial function to a sine wave would likely end in over-fitting, as it is overly complicated for the target function and would start fitting to the noise. This can be seen by the red line when no regularization was

added, the line starts fitting to the noise rather than approximating the smooth curve of the sine wave. The black line shows the fit when regularization (with

a learning rate of 5) is added. This new fit approximates the target function with greater accuracy as when it was trained it was penalized for over-fitting and was forced to be smoother. Although the model chosen to approximate the

target function was overly complicated for the sine function itself, thus reducing the chance of it finding the exact sine wave, the regularization has helped to improve the fit

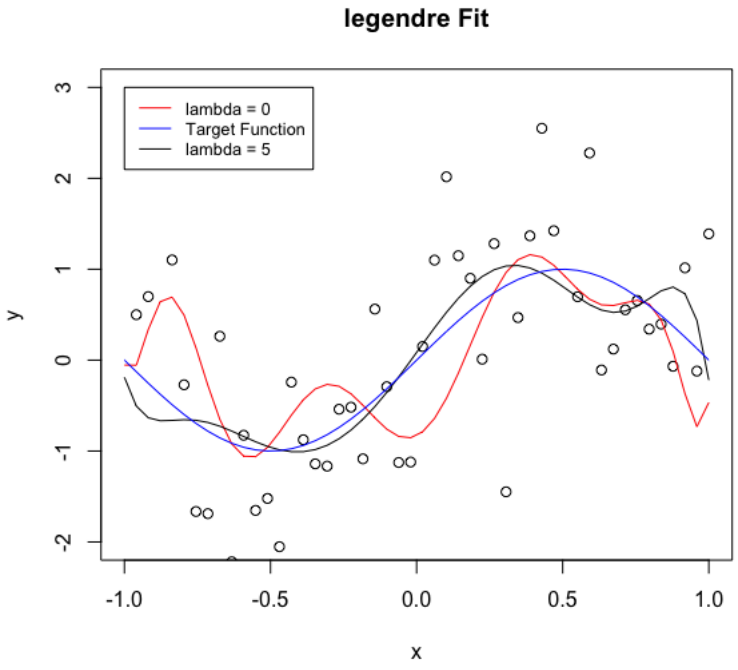

Figure 4: Legendre fits

2.3

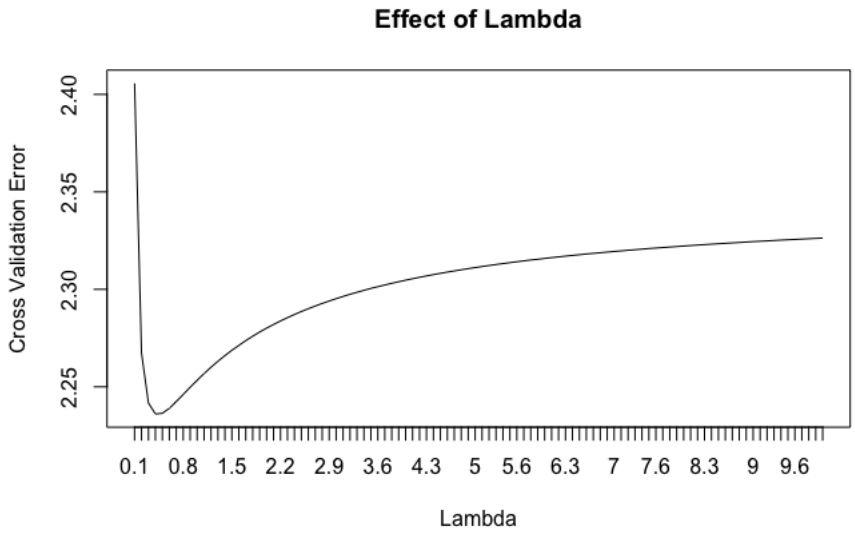

Figure 5: Effect of lambda

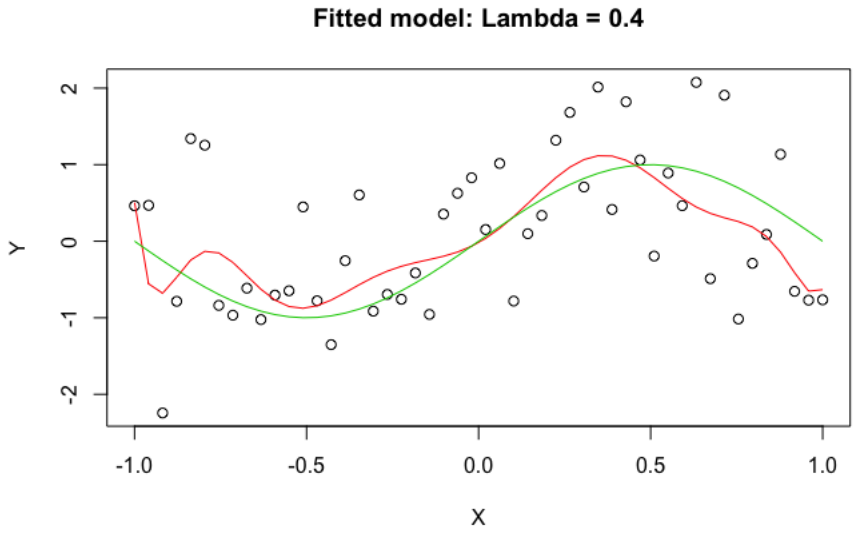

Figure 6: Fitted model with optimal lambda

3 Question 3

3.1

The figures below show in order from left to right: the average(mean) face of the images, the standard deviation face of the images, the original image and the scaled image

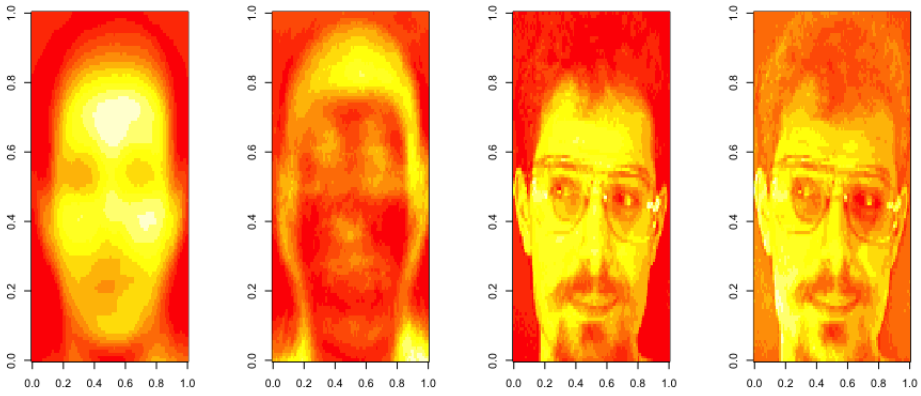

Figure 7: Mean, Std Deviation, Actual and Scaled images

3.2

The figures below show the first ten eigen-faces. The first image contains the eigen-face which represents the most information about the faces, the following faces to the right represent decreasing amounts of information about the variation of the original images.

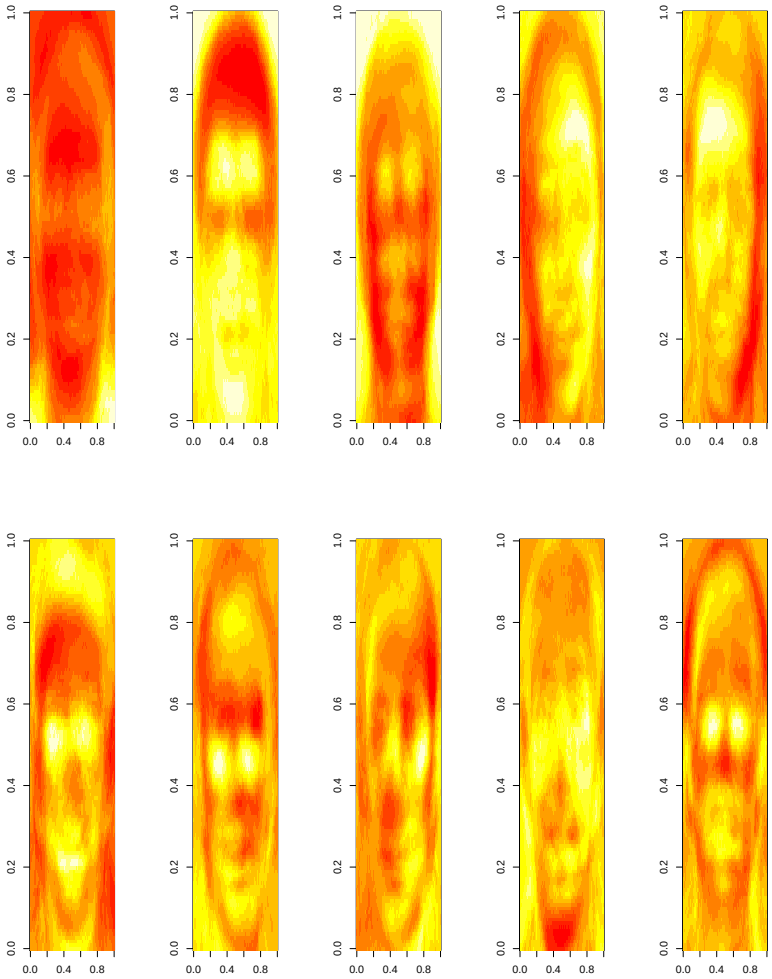

Figure 8: First ten Eigen-faces

3.3

The figures below in order from left to right show the scaled image for which I have attempted to reconstruct followed by the reconstruction of the face using 5 , 50 and 200 eigen-faces. The face reconstructed with 5 eigen-faces is indistinguishable. The face reconstructed with 50 eigen-values shows most of the main features of the original image, some of the features are quite blurred however it contains most of the essential information of the initial face as it can be recognized from the reconstructed image. The face reconstructed with 200 eigen-vectors is very close to the original. The blurred out features have become a lot more clear. From this we can see that around 50 eigen-vectors are needed to be able to recognize a face, and so the features which can be used to distinguish between individuals has been dramatically reduced, simplifying the classification problem.

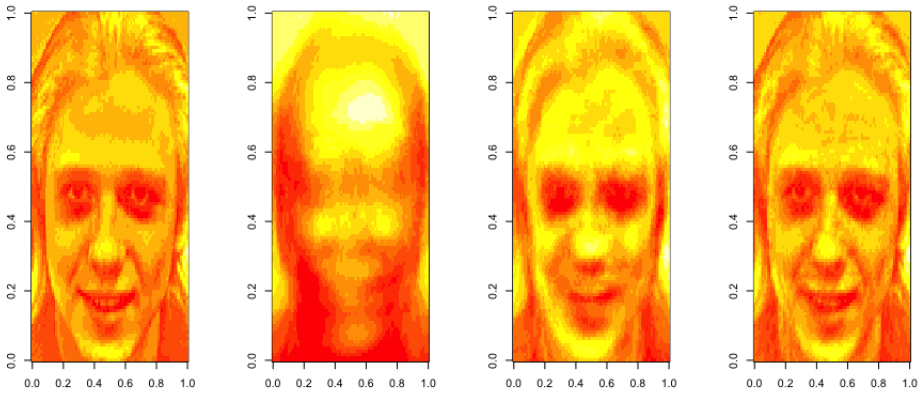

Figure 9: Original Face, Reconstructed Faces(5,50,200 eigen-faces)

4 R Code: Question 1

```
## Question 1.1
x <- seq(-1, 1, length.out = 30)

y = x*0.8 +rnorm(30,0,1)

fit1 = lm(y ~ 0+x, offset = rep(0.5, 30))
print(fit1$coefficients)
fit2 = lm(y ~ 0+x, offset = rep(-0.5, 30))

b1 = 0.6244
b2 = 0.6244

plot(x, x*b1 +0.5, col = "blue", type = 'l', ylim = c
    (-2,2), main = "Linear fits", ylab = "y")
lines(x, x*b2 -0.5, col = "red")
lines(x, x*0.8, col = "green")
points(x,y)
legend(-1,2, legend=c("Linear fit (0.5 intercept)", "
    Target Function", "Linear fit (-0.5 intercept)"),
        col=c("blue", "green", "red"), lty = c(1,1,1), cex
            =0.8)

## Question 1.2

generate = function(x)
{
  y = x*0.8 +rnorm(length(x),0,1)
  return(y)
}

generate.two = function(x)
{
  y = x*0.8
  return(y)
}

integrate.sum <- function(x,y,increment)
{
  sum <- 0

  for (i in 1:length(x))
  {
    sum <- sum + (x[i] - y[i])^2
```

```
  }
  return(abs(sum)*(increment))
}

val.i.one = rep(0,21)
val.i.two = rep(0,21)
out.i.one = rep(0,21)


for (i in 1:10000)
{
  x = runif(30,-1,1)
  y = generate(x)

  for (j in 5:25)
  {
    sample.var = sample(1:30, j)
    xtrain = x[sample.var]
    ytrain = y[sample.var]
    xtest = x[-sample.var]
    ytest = y[-sample.var]

    fit1 = lm(ytrain ~ 0 + xtrain, offset = rep(0.5,
        length(xtrain)))
    fit2 = lm(ytrain ~ 0 + xtrain, offset = rep(-0.5,
        length(xtrain)))

    predicted.one = fit1$coefficients*xtest + 0.5

    predicted.two = fit2$coefficients*xtest - 0.5


    val.one = sum((predicted.one-ytest)^2)/length(ytest)
    val.two= sum((predicted.two-ytest)^2)/length(ytest)
    val = max(c(val.one, val.two))
    val.i.one[j-5+1] = val.i.one[j-5+1] + val

    x.new = seq(-1,1, 0.01)
    y.true = generate.two(x.new)
    if(val.one> val.two)
    {
      v = fit1$coefficients*x.new + 0.5
    }
    else
    {
      v = fit2$coefficients*x.new - 0.5
```

```
    }


    bias <- (integrate.sum(v, y.true, 0.01))
    err.out <- bias + 1
    out.i.one[j-5+1] = out.i.one[j-5+1] + err.out
  }
}


plot.out = out.i.one/10000
plot.val = val.i.one/10000

par(mfrow = c(1,1))
plot(1:20, plot.out[20:1], type = 'l', col = 'blue', ylim
    = c(1.4,2.4), ylab = "Error", xlab = "Validation set
    size", main = "Validation error vs Expected error",
    xaxt = "n")
axis(1, at=1:20, labels=5:24)
lines(1:20, plot.val[20:1], type = 'l', col = 'red')
```

5 R Code: Question 2

```
##2.1

x = seq(-1, 1, length.out = 50)

sin.gen = function(x)
{
  y = sin(x*pi) + rnorm(length(x),0,1)
  return (y)
}

y = sin.gen(x)
plot(x,y, main = "Target Function")
lines(x, sin(x*pi), type = 'l', col = 2)

##2.2

sum.leg=function(x,y,pars,n,lam){
  val=0
  for(i in 1:n){
    val=val+(Legendre(x,i)*pars[i])
  }
  N = length(x)
  error = (val - y)^2
  E1 = sum(error)/N
  E2 = E1 + lam/N*(sum(pars)^2)
  return(list(out = val, E1 = E1, E2 = E2, coefficients =
      pars))
}

Legendre=function(x,n){
  val=0
  for(i in 0:n){
    val=val+((x^i)*choose(n,i)*choose((n+i-1)/2,n))
  }
  return((2^n)*val)
}

pars = runif(10, -1, 1)
Xtrain = seq(-1, 1, length.out = 50)
Ytrain = sin.gen(Xtrain)

lam = 0
```

```
obj = function(pars)
{
  res = sum.leg(Xtrain,Ytrain,pars,10,lam)
  return(res$E2)
}

res = nlm(obj,runif(10,-1,1),iterlim = 500)

fit =  sum.leg(Xtrain,Ytrain,res$estimate,10,lam)

lines(fit$out~Xtrain,col =2, pch = 16)

lam = 5

obj = function(pars)
{
  res = sum.leg(Xtrain,Ytrain,pars,10,lam)
  return(res$E2)
}

res = nlm(obj,runif(10,-1,1),iterlim = 500)

fit =  sum.leg(Xtrain,Ytrain,res$estimate,10,lam)

lines(fit$out~Xtrain,col =2, pch = 16)


##2.3

x = seq(-1, 1, length.out = 50)
folds <- cut(seq(1,length(x)),breaks=10,labels=FALSE)

y = sin.gen(x)
plot(x,y)

lambda_seq = seq(0.1, 10, 0.1)
cross.scores = rep(0, length(lambda_seq))

X = seq(-1, 1, length.out = 50)
Y = sin.gen(X)

for (i in 1:length(lambda_seq))
{
  pars = runif(10, -1, 1)
  lam = lambda_seq[i]
```

```
  val.sum = 0
  for(j in 1:10){

    testIndexes <- which(folds==j,arr.ind=TRUE)
    Xtest <- X[testIndexes]
    Ytest <- Y[testIndexes]
    Xtrain <- X[-testIndexes]
    Ytrain <- Y[-testIndexes]

    obj = function(pars)
    {
      res = sum.leg(Xtrain,Ytrain,pars,10,lam)
      return(res$E2)
    }

    res = nlm(obj,runif(10,-1,1),iterlim = 500)

    fit =  sum.leg(Xtrain,Ytrain,res$estimate,10,lam)

    predicted.ten=0
    for(l in 1:10){
      predicted.ten=predicted.ten+(fit$coefficients[l]*(
          Xtest^(l-1)))
    }

    val.error = sum(abs(predicted.ten - Ytest)^2)/length(
        Ytest)
    val.sum = val.sum + val.error
  }

  cross.scores[i] = val.sum
}


plot(1:length(lambda_seq), cross.scores/10, type = 'l',
    ylab = "Cross Validation Error", xlab = "Lambda", xaxt
    = "n", main = "Effect of Lambda")
axis(1, at=1:100, labels=lambda_seq)

##

lam = 0.4
pars = runif(10, -1, 1)
Xtrain = seq(-1, 1, length.out = 50)
Ytrain = sin.gen(Xtrain)
plot(Xtrain, Ytrain, main = "Fitted model: Lambda = 0.4",
```

```
     ylab = "Y", xlab = "X")

obj = function(pars)
{
  res = sum.leg(Xtrain,Ytrain,pars,10,lam)
  return(res$E2)
}

res = nlm(obj,runif(10,-1,1),iterlim = 500)

fit =  sum.leg(Xtrain,Ytrain,res$estimate,10,lam)

lines(fit$out~Xtrain,type = "l",col =2, pch = 16)
lines(sin(Xtrain*pi)~Xtrain, type = "l", col =3, pch =
    16)
```

6 R Code: Question 3

```
rm(list = ls())
library(pixmap)

##3.1

##using coding example to pull in the images

x=read.pnm(file = "Faces/1.pgm")

m  = prod(x@size)

n  = 400
xx =t(matrix(rev(x@grey),x@size[1],x@size[2]))
image(xx)
dims = x@size


X = matrix(0,n,prod(x@size))
for(i in 1:n)
{
  x=read.pnm(file = paste0("Faces/",i,".pgm"))
  X[i,] = rev(x@grey)
}

par(mfrow = c(1,4))

average = colMeans(X)
aa = t(matrix((average),x@size[1],x@size[2]))
image(aa)
flip.aa = aa

std.dev = apply(X,2,sd)
ss = t(matrix(std.dev,x@size[1],x@size[2]))
image(ss)



xx = t(matrix(X[168,],x@size[1], x@size[2]))
image(xx)

xx = t(matrix((X[168,]-average),x@size[1], x@size[2]))

scaled = xx /ss
```

```
image(scaled)

for (i in 1:n)
{
  X[i,] = (X[i,] - average)/std.dev
}


## 3.2

A = (X)%*%t(X)

A= A/400

eigen.var = eigen(A)

eigen.faces = t(X)%*%eigen.var$vectors

par(mfrow = c(2,5))

for (i in 1:10)
{
  xx = t(matrix(eigen.faces[,i],x@size[1], x@size[2]))
  image(xx)
}

## 3.3

par(mfrow = c(1,4))

eigen.diag = diag(1/(eigen.var$values)^0.5)

eigen.scaled = eigen.faces%*%eigen.diag

xx = t(matrix(X[115,],x@size[1], x@size[2]))
image(xx)

no.faces = c(5,20,200)

for (j in no.faces)
{
  start = X[115,]

  reconstruct = t(eigen.scaled[,1])%*%start
```

```
  sum = reconstruct%*%eigen.scaled[,1]

  for (i in 2:j)
  {
    reconstruct = t(eigen.scaled[,i])%*%start
    recon = reconstruct%*%eigen.scaled[,i]
    sum = sum + recon
  }

  xx = t(matrix(sum, x@size[1], x@size[2]))
  image(xx)
}
```