Michael Leibert

Math 504

Final Exam

1. This problem aims to review several ideas - the $Ax = b$ problem, penalty methods, quadratic optimization, basis functions, positive definite matrices, projection, orthogonality - within the context of Fourier transforms. The essential idea is that every function $f(x)$ with domain on some interval $[0, L]$ can be written in terms of an infinite number of cosine basis functions as follows:

$$f(x) = \alpha_0 + \sum_{j=1}^{\infty} \alpha_j \cos\left(\frac{2\pi j x}{L}\right)$$

where $x \in [0, L]$. Sauer alludes to this in section 11.1 where he discusses the discrete cosine transform. If you want to read more, the attached notes of David Wilkins (especially see sections 8.2 and 8.4) are nice.

In this problem, we will not deal with an infinite expansion, but will instead consider a basis composed of n of the cosine basis functions. Consider, as in previous homeworks, the female samples of the BoneMass dataset. Let N be the number of samples, i.e $N = 259$. Let $b_j(x) = \cos(2\pi j(x - x_{\min})/(x_{\max} - x_{\min}))$ where $x_{\min}$ and $x_{\max}$ are the minimum and maximum age values in the dataset. Let $b_0(x) = 1$.

$$f(x) = \sum_{j=0}^{n} \alpha_j b_j(x)$$

(a) Consider the basis functions $b_0(x), b_1(x), \dots, b_5(x)$ and let $\mathcal{F}$ be the linear function space spanned by these 6 functions. We have discussed orthonormal vectors, but we have not discussed orthonormal functions. To do so, we need to introduce a norm and a dot-product on $\mathcal{F}$. The most common such dot-product is formed by integrating two functions against each other. Specifically, let $g(x), h(x) \in \mathcal{F}$. Then define the dot product, written $< g, h >$ by

$$< g, h >= \int_{x_{\min}}^{x_{\max}} g(x)h(x)dx,$$

and the associated norm given by $\|g\|^2 =< g, g >$. This norm is known as the L^2 norm. Using this L^2 norm and dot-product, use numerical integration to show that the $b_i(x)$ are orthogonal and compute the coefficients necessary to normalize them. Write your own numerical integrator. Using numerical integration, you will find that $< b_i, b_j >$ for $i \neq j$ is small but not 0 due to round-off error. Explain why the value you get is small enough to be attributed to round-off error.

```r
rm(list = ls()); setwd("G:\\math\\504");options(scipen=999)
bones<-read.table("BoneMassDataF.txt",header=T)
bones<-bones[which(bones[,3] == "female"),]

Fapprox<-function(m,n=10000,x){

  MATT<-matrix(0:(m-1),m,1)

  Gx<-function(v,x){ if(v != 0) {
    return( cos( (2*pi*v*(x-9.4 ) ) /(25.55-9.4 )) )
    } else  {return(rep(1,length(x) ))} }

  a=min(x);b=max(x)
  h=(b-a)/n;i=0:(n-1)
  W<-apply(MATT,1,function(w) Gx(w,a+(i+1)*h) )
  Mat<-matrix(NA,m,m)

  for( i in 1:m){
    Mat[,i] <-   colSums( (  W[  ,i ]*W[  ,1:m]   ) ) * h }

return(Mat)
}

Fapprox(6,n=10000, bones$age)
```

```
##                          [,1]                      [,2]
## [1,] 16.149999999999985789145 -0.000000000000002216161
## [2,] -0.000000000000002216161  8.074999999999992894573
## [3,] -0.000000000000002057479 -0.000000000000002262779
## [4,] -0.000000000000002003689 -0.000000000000001754460
## [5,] -0.000000000000001406168 -0.000000000000002019826
## [6,] -0.000000000000002067341 -0.000000000000002088857
##                          [,3]                      [,4]
## [1,] -0.000000000000002057479 -0.000000000000002003689
## [2,] -0.000000000000002262779 -0.000000000000001754460
## [3,]  8.074999999999992894573 -0.000000000000001924796
## [4,] -0.000000000000001924796  8.074999999999992894573
## [5,] -0.000000000000002312535 -0.000000000000001626260
## [6,] -0.000000000000001361791 -0.000000000000001532127
##                           [,5]                       [,6]
## [1,] -0.0000000000000014061682 -0.0000000000000020673407
## [2,] -0.0000000000000020198260 -0.0000000000000020888569
## [3,] -0.0000000000000023125349 -0.0000000000000013617912
## [4,] -0.0000000000000016262602 -0.0000000000000015321272
## [5,]  8.0749999999999928945726 -0.0000000000000002815026
## [6,] -0.0000000000000002815026  8.0749999999999928945726
```

We can see from out output matrix that the $b_i(x)$'s are orthogonal, but not orthonormal.

```
Gapprox<-function(n,j,k,x){

    coeff<-(1/sqrt( diag(Fapprox(6,n=10000, bones$age)) ))
    Fx<-function(l,x){ if(l != 0) {
        return( coeff[l+1]* cos( (2*pi*l*(x-9.4 ) ) /(25.55-9.4 )) )
        } else  {return( coeff[l+1]* rep(1,length(x) ))} }

    a=min(x);b=max(x)

    h=(b-a)/n;i=0:(n-1)
    return(     sum( Fx(j,a+(i+1)*h)*Fx(k,a+(i+1)*h)*h )    )
}

matt<-matrix(NA,6,6)
for( J in 0:5){for( K in 0:5){ matt[J+1,K+1]<-Gapprox(10000,J,K,bones[,2]) }}
matt;diag(matt )
```

```
##                           [,1]                       [,2]
## [1,]  1.0000000000000044408921 -0.0000000000000001931235
## [2,] -0.000000000000000001931235  0.999999999999999988897770
## [3,] -0.000000000000000001764539 -0.000000000000000002870425
## [4,] -0.000000000000000001656119 -0.000000000000000002258529
## [5,] -0.000000000000000001279359 -0.000000000000000002429968
## [6,] -0.000000000000000001767250 -0.000000000000000002429290
##                            [,3]                        [,4]
## [1,] -0.000000000000000001764539 -0.000000000000000001656119
## [2,] -0.000000000000000002870425 -0.000000000000000002258529
## [3,]  0.999999999999999988897770 -0.000000000000000002406251
## [4,] -0.000000000000000002406251  0.999999999999999988897770
## [5,] -0.000000000000000002953096 -0.000000000000000002110128
## [6,] -0.000000000000000001658829 -0.000000000000000001872282
##                             [,5]                         [,6]
## [1,] -0.0000000000000000012793586 -0.0000000000000000017672495
## [2,] -0.0000000000000000024299681 -0.0000000000000000024292905
## [3,] -0.0000000000000000029530957 -0.0000000000000000016588293
## [4,] -0.0000000000000000021101285 -0.0000000000000000018722816
## [5,]  0.9999999999999999888977698 -0.0000000000000000005817422
## [6,] -0.0000000000000000005817422  0.9999999999999999888977698
```

```
## [1] 1 1 1 1 1 1
```

We have computed the coefficients necessary to normalize them.

Error for Riemann integration:

$$\begin{aligned}
\text{error} &= \left| \int_a^{a+h} f(x)\ \mathrm{d}x - f(a)\cdot h \right| \\
&\approx \left| \int_a^{a+h} \Big(f(a) + f'(a)(x-a) \Big)\ \mathrm{d}x - f(a)\cdot h \right| \\
&= \left| f(a)\cdot h + f'(a)\frac{(x-a)^2}{2}\ \Big|_a^{a+h} - f(a)\cdot h \right| \\
&= f'(a)\frac{h^2}{2} \\
&\approx h^2 \text{ (error for one rectangle)}
\end{aligned}$$

Riemann integration has an error of roughly $\left(h^2\right)\left(\dfrac{b-a}{h}\right) = (b-a)\cdot h$, where h is the grid width.

The total error is about 0.02608486 so we can attribute the error we are getting to roundoff error.

(b) Continuing from (a), find $f(x) \in \mathcal{F}$ that best fits the data using least squares as the loss function. Justify each step in your computation. Plot the data and resulting fit. Explain how determining $f(x)$ corresponds to (i) a linear regression (ii) projection of a vector onto a linear space (iii) quadratic optimization.

In this step, I am writing a function for any general b_j. If w is 1,...,5, I obtain the normalizing coefficients, apply them to the $b_j = 1, ..., 5$, and return the specified b_j. If 0 is given I return 1's.

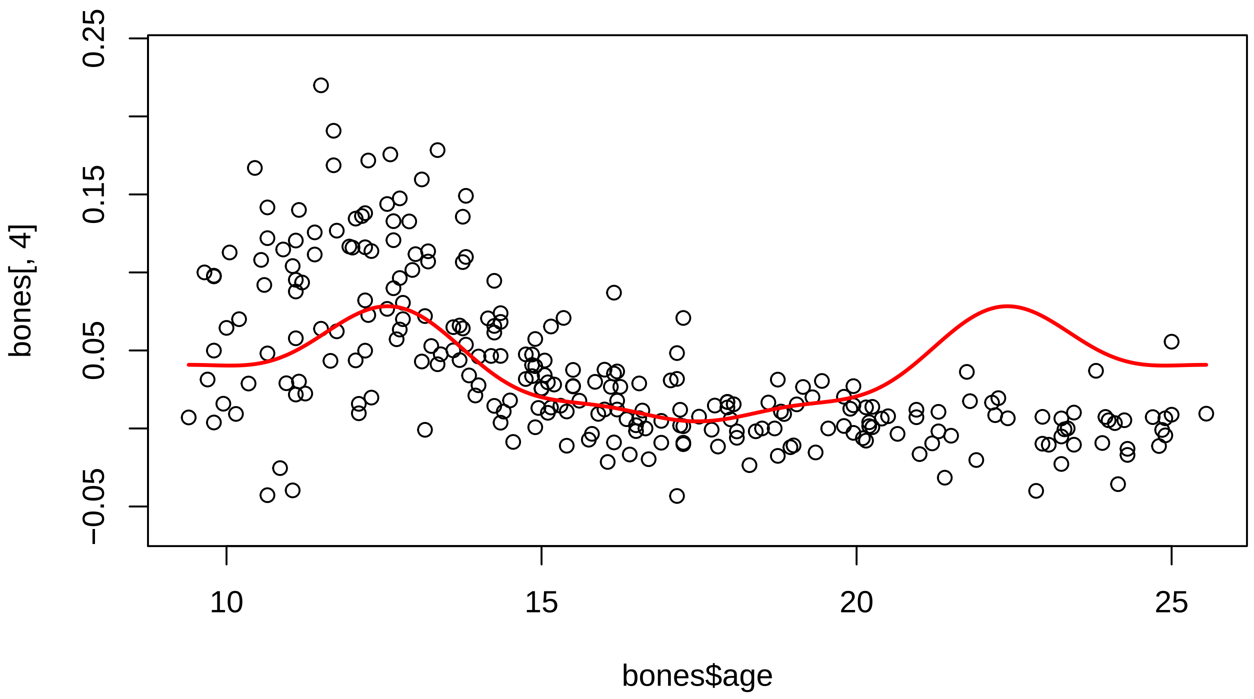
```
Fx<-function(w,x){
    #need this matrix for the normalizing Coefficients
    ncoef<-(1/sqrt( diag(Fapprox(6,n=10000, bones$age)) ))
    if(w == 0 ) {return(   rep(1,length(x) ))  } else {
    return( ncoef[w+1]* cos( (2*pi*w*(x-9.4 ) ) /(25.55-9.4 )) )
        } }
```

In this step I form the model matrix. By using our function Fx above. Then I solve for α.

```
model_matrix <- function(x,m) {
    nx <- length(x)
    A<-matrix(NA,nx,m+1)
    for( i in 0:m){     A[,(i+1)] <- Fx(i,x)  }
    colnames(A )<-NULL; return(A)   }
B<- model_matrix(bones$age,5)
Alpha <- solve(t(B) %*% B , t(B) %*% as.matrix(bones[,4]))
```

```
x_grid <- seq(min(bones$age), max(bones$age), .01)
B_grid <- model_matrix(x_grid,5); y_grid <- B_grid %*% Alpha
```

```
par(mar=c(4.1,4.1,2,1))
plot(bones$age, bones[,4], ylim=c(min(bones[,4])-.02,max(bones[,4]) + .02) )
lines(x_grid, y_grid, col="red", lwd=2)
```



We have some data, $x^{(i)} \in \mathbb{R}$ and $y_i \in \mathbb{R}$, and we wish to approximate y by a function $f(x)$, when $f(x) \in \mathcal{F}$ and $f(x) : \mathbb{R}^n \to \mathbb{R}$.

Let $\mathcal{F}$ be:

$$\mathcal{F} = \{f(x) : f(x) : \mathbb{R}^n \to \mathbb{R}, f(x) = 5^{th} \text{ harmonic}\}.$$

We use least squares to find the $f(x) \in \mathcal{F}$

$$\min_{f(x) \in \mathcal{F}} \sum_{i=1}^{N} \left| y_i - f\left(x^{(i)}\right) \right| \tag{1}$$

It is key to note that $\mathcal{F}$ is a linear function / vector space. It is true that for any $g(x) \in \mathcal{F}$ and $h(x) \in \mathcal{F}$, $c_1 g(x) + c_2 h(x) \in \mathcal{F}$, where $c_1, c_2 \in \mathbb{R}$

Our function $f(x)$ can be written as:

$$f(x) = \sum_{j=0}^{n} \alpha_j b_j(x) = \alpha_0(1) + \alpha_1 \cos\left(\frac{2\pi(1)\left(x - x_{max}\right)}{\left(x_{max} - x_{min}\right)}\right) + ... + \alpha_5 \cos\left(\frac{2\pi(5)\left(x - x_{max}\right)}{\left(x_{max} - x_{min}\right)}\right)$$

so we can rewrite (1) as:

$$\min_{f(x) \in \mathcal{F}} \sum_{i=1}^{N} \left| y_i - f\left(x^{(i)}\right) \right| = \min_{\alpha \in \mathbb{R}^6} \sum_{i=1}^{N} \left| \alpha_0 + \alpha_1 \cos\left(\frac{2\pi\left(x_i - x_{max}\right)}{\left(x_{max} - x_{min}\right)}\right) + ... + \alpha_5 \cos\left(\frac{10\pi\left(x_i - x_{max}\right)}{\left(x_{max} - x_{min}\right)}\right) \right|^2 . \tag{2}$$

The sum in the RHS of (2) can be put into matrix form:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} - \begin{pmatrix} b_0\left(x^{(1)}\right) & b_1\left(x^{(1)}\right) & ... & b_5\left(x^{(1)}\right) \\ b_0\left(x^{(2)}\right) & b_1\left(x^{(2)}\right) & ... & b_5\left(x^{(2)}\right) \\ \vdots & \vdots & \ddots & \vdots \\ b_0\left(x^{(N)}\right) & b_1\left(x^{(N)}\right) & ... & b_5\left(x^{(N)}\right) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{pmatrix} \tag{3}$$

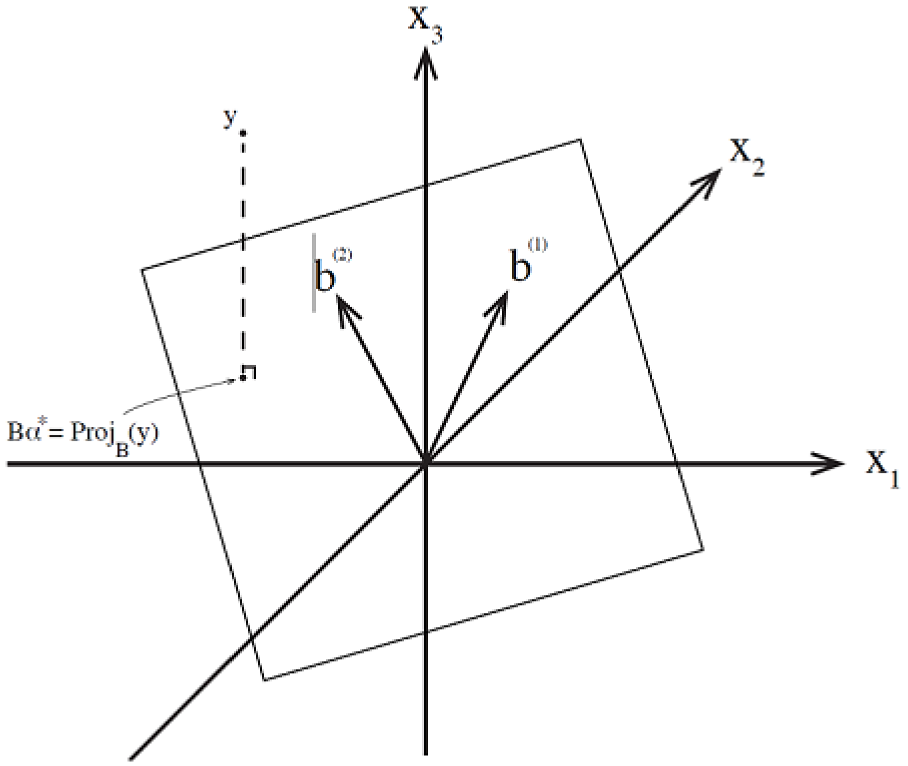
and (2) is equivalent to

$$\min_{\alpha \in \mathbb{R}^6} ||\boldsymbol{y} - B\boldsymbol{\alpha}||^2 . \tag{4}$$

So given some data we are trying to fit through that data the best 5^{th} harmonic. We are not doing regression in the sense of fitting a line because we are fitting a 5^{th} harmonic. But we end up doing the same exact thing that we do for linear regression (minimize sum of squares, replace with model matrix).

The important part of linear regression is not that the functions are linear, it is that the space of the functions we are considering are a linear function space. We are still linear in the parameters. Thus in this context, determining $f(x)$ corresponds to a linear regression.

Now for projection consider the $\boldsymbol{y}$, B and $\boldsymbol{\alpha}$ from (3) above. We wish to choose $\boldsymbol{\alpha}$ so that $B\boldsymbol{\alpha}$ is as close as possible to $\boldsymbol{y}$. Or put another way, find the point in the Span$(B) = \text{Span}\begin{pmatrix} b^{(0)} & b^{(1)} & \dots & b^{(5)} \end{pmatrix}$ closest to $\boldsymbol{y}$. If $n = 2$, we would have a visual like this:



For $n = 6$, b_0 through b_5 are linearly combining to form a plane Again, we want to find the point closest to $\boldsymbol{y}$. It will be the point that is orthogonal and this point will be the $\text{Proj}_B(y)$.

The projection of $x \in \mathbb{R}$ onto Ω is the closest point in Ω to x.

$$P_\Omega(x) = \min_{z \in \Omega} ||z - x||$$

Now suppose $\Omega = \text{Span}\begin{pmatrix} b^{(0)} & b^{(1)} & \dots & b^{(5)} \end{pmatrix}$, and $P_\Omega(x) = \min_{z \in \Omega} ||z - y||$, where

$$z = \alpha_0 b^{(0)} + \alpha_1 b^{(1)} + ... + \alpha_5 b^{(5)} = B\boldsymbol{\alpha} = \begin{pmatrix} b^{(0)} & b^{(1)} & \dots & b^{(5)} \end{pmatrix} \begin{pmatrix} \alpha_0 & \alpha_1 & \dots & \alpha_5 \end{pmatrix}^T$$

and

$$\min_{z\in\Omega} ||z - y|| = \min_{z\in\Omega} ||\alpha_0 b^{(0)} + \alpha_1 b^{(1)} + ... + \alpha_5 b^{(5)} - y|| \tag{5}$$

So we have $\min\limits_{\alpha\in\mathbb{R}} ||B\alpha - y||^2$. To tie projection and linear regression together, we can view linear regression as a projection onto the span of the columns of the model matrix.

Now tying both of them to quadratic optimization. When we have a linear subspace and project onto it, it leads to a minimization of a quadratic $\min\limits_{\alpha\in\mathbb{R}^6} ||B\boldsymbol{\alpha} - \boldsymbol{y}||^2$, (5) above. When $\mathcal{F}$ is a linear function space then the regression is a minimization of a quadratic $\min\limits_{\alpha\in\mathbb{R}^6} ||\boldsymbol{y} - B\boldsymbol{\alpha}||^2$, (4) above.

Both (4) and (5) are solved with the normal equations.

$$\begin{aligned}
L(\alpha) &= \sum_{i=1}^{N} \left(y_i - \alpha_0 + \alpha_1 x_1^{(i)} + \alpha_2 x_2^{(i)} + ...\alpha_n x_n^{(i)} \right)^2 \\
&= \sum_{i=1}^{N} r_i^2 \\
&= \mathbf{r}\cdot\mathbf{r} \\
&= (y - B\alpha)\cdot(y - B\alpha) \\
&= (y - B\alpha)^T (y - B\alpha) \\
&= \left(y^T - (B\alpha)^T\right)(y - B\alpha) \\
&= y^T y - (B\alpha)^T y - y^T B\alpha + (B\alpha)^T B\alpha \\
&= y^T y - 2(B^T y)^T \alpha + \alpha^T B^T B\alpha
\end{aligned}$$

$$\begin{aligned}
\nabla L(\alpha) &= -\frac{1}{2}\left(B^T B\right)^{-1}\left(-2B^T y\right) \\
&= \left(B^T B\right)^{-1} B^T y
\end{aligned}$$

We have shown that determining $f(x)$ corresponds to a linear regression, projection of a vector onto a linear space, and a quadratic optimization.

(c) Consider $\mathcal{F}$, but now formed from all $b_i(x)$ with $i = 0, 1, \ldots, 1000$. Show that the best fit is now not unique. Construct a penalty term that penalizes $f(x)$ (or equivalently α) for contributions from $b_j(x)$ with large j. Show that your penalty term creates a unique best fit $f(x)$ for any penalty value $\rho > 0$. (Recall, you can do this by showing that the penalty term corresponds to a positive definite matrix. Go through the argument). What happens to the fit as ρ increases? Show plots of the data and fit for different ρ.

We have an underdetermined system, more variables than equations. There cannot be more basic variables than there are equations, so there must be at least one free variable. Such a variable may be assigned infinitely many different values. Because our system is consistent (has at least one solution), each different value of a free variable will produce a different solution.

We note, $\alpha = \left(B^T B + \rho\ \Omega\right)^{-1}$ is a unique minimum. If ρ is 0, $B^T B$ will not be invertible because B is a fat matrix and there are some values B takes to 0.

If $\rho > 0$ we get a unique solution and $B^TB + \rho\Omega$ can be inverted. Both are B^TB and $\rho\ \Omega$ are positive semidefinite matrices.

$$v^T(B^TB)v \geq 0 \text{ and } v^T(\rho\ \Omega)v \geq 0$$

For any α, $\alpha^TB^TB\alpha = ||B\alpha||^2 > 0$. We also know,

$$\alpha^T\Omega\alpha = \int_{x_{min}}^{x_{max}} \left(\left[\sum_j \alpha_j b_j(z) \right]'' \right)^2 \mathrm{d}z.$$

The integral is non-negative since the integrand is square, so $\alpha^T\Omega\alpha \geq 0$. The only way $\alpha^T\Omega\alpha = 0$ is if

$$\sum_j \alpha_j b_j(z)$$

has a second derivative that is zero for all z, that is the function is linear.

If α is such that $\sum\limits_j \alpha_j b_j(z)$ is not linear, then $\alpha^T\Omega\alpha > 0$, and we can conclude $\alpha^t \left(B^TB + \rho\Omega\right)\alpha > 0$, which in turn gives $\left(B^TB + \rho\Omega\right)\alpha \neq 0$.

If α is such that $\sum\limits_j \alpha_j b_j(z)$ is linear, then $\alpha^T\Omega\alpha = 0$. However, the ith coordinate of the vector $B\alpha$, is given by

$$\sum_j \alpha_j b_j(x_i),$$

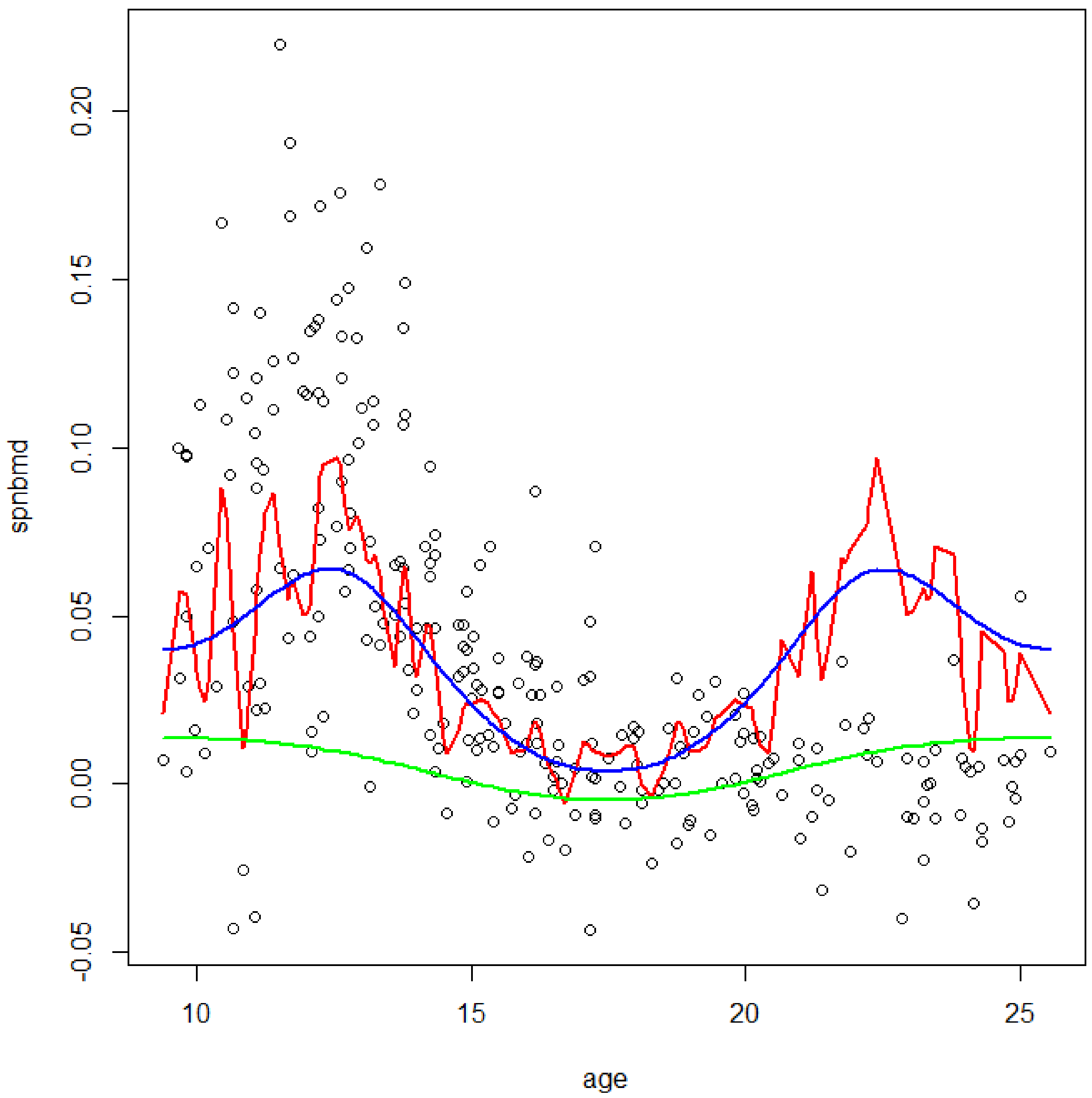
where x_i is the ith sample. If $B\alpha = 0$, then the function $\sum\limits_j \alpha_j b_j(z)$ is a linear function that is zero at all x_i. But this means the function is zero and hence $\alpha = 0$.

This allows us to conclude that if $\alpha^T\Omega\alpha = 0$ and $\alpha \neq 0$, then $B\alpha \neq 0$ and therefore $\alpha^TB^TB\alpha > 0$. We can also conclude that in this case $\left(B^TB + \rho\Omega\right)\alpha \neq 0$.

Therefore, for either of the two cases when $\sum_j \alpha_j b_j(z)$ is linear or not linear, we have shown that for any $\alpha \neq 0$ we have $\left(B^TB + \rho\Omega\right)\alpha \neq 0$, so $B^TB + \rho\ \Omega$ is invertible.

```r
Fx<-function(w,x){
    #need this matrix for the normalizing Coefficients
    ncoef<-(1/sqrt( diag(Fapprox(1001,n=10000, bones$age)) ))
    if(w == 0 ) {return(   rep(1,length(x) ))  } else {
```

```r
    return( ncoef[w+1]* cos( (2*pi*w*(x-9.4 ) ) /(25.55-9.4 )) )
        } }
dmat<-function(m,n=10000,x){

MATT<-matrix(0:(m-1),m,1)

dFx<-function(v,x){ if(v != 0) {
    return( -((4*pi^2*v^2)/(16.15^2))* cos( (2*pi*v*(x-9.4 ))/16.15)
    )   } else  {return(rep(1,length(x) ))} }

a=min(x);b=max(x)
h=(b-a)/n;i=0:(n-1)
W<-apply(MATT,1,function(w) dFx(w,a+(i+1)*h) )
}

Bpp<-dmat(1000+1,n=10000, bones$age)
z<-seq(min(bones$age),max(bones$age),   length.out = 10000 )
h<-z[2]-z[1]


omega<-h*(t(Bpp) %*% Bpp )  #Penalty Term

B<- model_matrix(bones$age,1000)
n=1001
y<-as.matrix( bones[,4] )
A<-matrix(NA,(n ),3)
A[,1]<-solve( t(B)%*%B + (.0001 * omega) ) %*% t(B)%*% y
A[,2]<-solve( t(B)%*%B + (1 * omega) ) %*% t(B)%*% y
A[,3]<-solve( t(B)%*%B + (100 * omega) ) %*% t(B)%*% y


par(mar=c(4.1,4.1,1,1))
datt<-data.frame(bones$age, B%*% A[,1], B%*% A[,2], B%*% A[,3] )
datt<-datt[with(datt, order(bones$age)), ]

plot(bones[,2], bones[,4], xlab="age", ylab="spnbmd")
lines( datt[,1],datt[,2],col="red", lwd=2)
lines( datt[,1],datt[,3],col="blue", lwd=2)
lines( datt[,1],datt[,4],col="green", lwd=2)
```



As ρ increases the regression becomes more linear in order to reduce the penalty term, the result is that the fitting term increases, meaning that the fit becomes poorer.

(2) This problem repeats the neural net problem of hw 12, but using Martens' conjugate gradient method. You will apply his Algorithm 1, although with some differences. As we discussed in class, let $f(\eta)$ be the **negative** log-likelihood function (negative, so we can minimize rather than maximize). Martens' uses θ for the net parameters, but to connect to our notation, I will use η.

(a) In Algorithm 1, Martens defines $B = Hf(\eta) + \lambda I$. Explain why Martens adds a λI. (This connects to our discussion of Hessian modification.) Martens changes λ at every iteration of the optimization as he discusses in Section 4.1. Here, let's take a simplified approach. Let $\eta^{(i)}$ and $\eta^{(i+1)}$ be the ith and $i+1$th values of η in the optimizatio iteration. If the algorithm has made good progress in iteration $i+1$ relative to $\eta^{(i)}$, then lower λ. If the algorithm has made poor progress, then raise λ. It is up to you to define good/poor and to determine how much to lower/raise λ. Explain why we should raise λ if the progress is poor.

The neural net is a nonconvex optimization, we may go to a local minimum. We want the hessian of the negative log likelihood to be positive definite, or alternatively we want all the eigenvalues to be positive. However, for neural nets the hessian of negative log likelihood is not going to be positive definite because it is not a convex function.

However we can modify the hessian such that it is positive definite and similar to the hessian. We can rewrite the hessian as a decomposition, and add a constant so the minimum eigenvalue becomes positive

$$H = Q \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} Q^T$$

$$H + \lambda I = Q \begin{pmatrix} \lambda_1 + \lambda & & & \\ & \lambda_2 + \lambda & & \\ & & \ddots & \\ & & & \lambda_n + \lambda \end{pmatrix} Q^T$$

$$= Q \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} Q^T + Q \begin{pmatrix} \lambda & & & \\ & \lambda & & \\ & & \ddots & \\ & & & \lambda \end{pmatrix} Q^T.$$

We should raise λ if we are doing poorly, because raising the lambdas enough will get all of our eigenvalues positive. When the eigenvalues are all positive we are in a region of the function that is locally convex and we will go down.

(b) Martens' points out that conjugate gradient does not require actually computing $Hf(\eta)$, but rather being able to compute $Hf(\eta)d$ for any vector d. He then notes

$$Hf(\eta)d = \lim_{\epsilon \to 0} \frac{\nabla f(\eta + \epsilon d) - \nabla f(\eta)}{\epsilon}$$

Prove this relation. (Hint: Consider the first coordinate of this equality, which is given by

$$[Hf(\eta)d]_1 = \lim_{\epsilon \to 0} \frac{\frac{\partial f}{\partial \eta_1}(\eta + \epsilon d) - \frac{\partial f}{\partial \eta_1}(eta)}{\epsilon},$$

where $[Hf(\eta)d]_1$ is the first coordinate of the vector $Hf(\eta)d$. Write a first-order, multi-d Taylor series expansion for $\frac{\partial f}{\partial \eta_1}(\eta + \epsilon d)$ about the base point η. Plug the Taylor series in, and show that the limit converges to $[Hf(\eta)d]_1$. Other coordinates will follow by the same argument.

First, let's write down $Hf(\eta)d$

$$Hf(\eta)d = \begin{pmatrix} \frac{\partial^2 f(\eta)}{\partial \eta_1^2} & \frac{\partial^2 f(\eta)}{\partial \eta_2 \partial \eta_1} & \cdots & \frac{\partial^2 f(\eta)}{\partial \eta_n \partial \eta_1} \\ \frac{\partial^2 f(\eta)}{\partial \eta_2 \partial \eta_1} & \frac{\partial^2 f(\eta)}{\partial \eta_2^2} & \cdots & \frac{\partial^2 f(\eta)}{\partial \eta_n \partial \eta_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\eta)}{\partial \eta_n \partial \eta_1} & \frac{\partial^2 f(\eta)}{\partial \eta_n \partial \eta_2} & \cdots & \frac{\partial^2 f(\eta)}{\partial \eta_n^2} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix} \tag{6}$$

$$= \begin{pmatrix} \frac{\partial^2 f(\eta)}{\partial \eta_1^2} d_1 + \frac{\partial^2 f(\eta)}{\partial \eta_2 \partial \eta_1} d_2 + \ldots + \frac{\partial^2 f(\eta)}{\partial \eta_n \partial \eta_1} d_n \\ \frac{\partial^2 f(\eta)}{\partial \eta_2 \partial \eta_1} d_1 + \frac{\partial^2 f(\eta)}{\partial \eta_2^2} d_2 + \ldots + \frac{\partial^2 f(\eta)}{\partial \eta_n \partial \eta_2} d_n \\ \vdots \\ \frac{\partial^2 f(\eta)}{\partial \eta_n \partial \eta_1} d_1 + \frac{\partial^2 f(\eta)}{\partial \eta_n \partial \eta_2} d_2 + \ldots + \frac{\partial^2 f(\eta)}{\partial \eta_n^2} d_n \end{pmatrix} \tag{7}$$

$$= \begin{pmatrix} \nabla \left(\frac{\partial}{\partial \eta_1} f(\eta) \right) \cdot \boldsymbol{d} \\ \nabla \left(\frac{\partial}{\partial \eta_2} f(\eta) \right) \cdot \boldsymbol{d} \\ \vdots \\ \nabla \left(\frac{\partial}{\partial \eta_i} f(\eta) \right) \cdot \boldsymbol{d} \\ \vdots \\ \nabla \left(\frac{\partial}{\partial \eta_n} f(\eta) \right) \cdot \boldsymbol{d} \end{pmatrix} \tag{8}$$

Now we can show that

$$Hf(\eta)d = \lim_{\epsilon\to 0}\frac{\nabla f(\eta+\epsilon d)-\nabla f(\eta)}{\epsilon}$$

is equal to $Hf(\eta)d$. Consider the ith coordinate of the vector $Hf(\eta)d$.

$$\Big[Hf(\eta)d\Big]_i = \lim_{\epsilon\to 0}\frac{\frac{\partial}{\partial\eta_i}f(\eta+\epsilon\boldsymbol{d})-\frac{\partial}{\partial\eta_i}f(\eta)}{\epsilon} \tag{9}$$

First-order Taylor series expansion about the base point η:

$$\frac{\partial}{\partial\eta_i}f(\eta+\epsilon\boldsymbol{d}) \approx \frac{\partial}{\partial\eta_i}f(\eta)+\nabla\left(\frac{\partial}{\partial\eta_i}f(\eta)\right)\cdot\epsilon\boldsymbol{d}.$$

Plug back the expansion back into (9)

$$\begin{aligned}\Big[Hf(\eta)d\Big]_i &= \lim_{\epsilon\to 0}\frac{\frac{\partial}{\partial\eta_i}f(\eta+\epsilon\boldsymbol{d})-\frac{\partial}{\partial\eta_i}f(\eta)}{\epsilon}\\ &\approx \lim_{\epsilon\to 0}\frac{\cancel{\frac{\partial}{\partial\eta_i}f(\eta)}+\nabla\left(\frac{\partial}{\partial\eta_i}f(\eta)\right)\cdot\cancel{\epsilon}\boldsymbol{d}-\cancel{\frac{\partial}{\partial\eta_i}f(\eta)}}{\cancel{\epsilon}}\\ &= \nabla\left(\frac{\partial}{\partial\eta_i}f(\eta)\right)\cdot\boldsymbol{d}\end{aligned}$$

We see that for the ith coordinate is equal to the ith row in (8) above. Other coordinates will follow by the same argument.

(c) In your code, use a finite ϵ to approximate $Hf(\eta)d$:

$$Hf(\eta)d \approx \frac{\nabla f(\eta+\epsilon d)-\nabla f(\eta)}{\epsilon} \tag{10}$$

Your code from hw 12 already computes ∇f (remember though, now we are using the negative log likelihood). Reuse that code. Choose an appropriate value of ϵ and explain your choice. (Hint: Consider the error you make in computing the partials of f, then choose an ϵ that balances this error with the Taylor series error, as we did in analyzing finite differences in class.)

```
Hfd<-function(x,y,eta,m,d,epi=10^-3 ){  (gradL(x,y,eta+d*epi,m)-gradL(x,y,eta,m))/(epi)}
```

We consider the error made in computing the partials of f,

$$\begin{aligned}
\text{error} &= \frac{\frac{\partial}{\partial\eta}f(\eta+h\boldsymbol{d})-\frac{\partial}{\partial\eta}f(\eta)}{h}-[Hf(\eta)d] \\
&= \frac{\frac{\partial}{\partial\eta}f(\eta+h\boldsymbol{d})(1+\epsilon_1)-\frac{\partial}{\partial\eta}f(\eta)(1+\epsilon_2)}{h}-[Hf(\eta)d] \\
&= \frac{\frac{\partial}{\partial\eta}f(\eta+h\boldsymbol{d})\epsilon_1-\frac{\partial}{\partial\eta}f(\eta)\epsilon_2}{h}+\frac{\frac{\partial}{\partial\eta}f(\eta+h\boldsymbol{d})-\frac{\partial}{\partial\eta}f(\eta)}{h}-[Hf(\eta)d] \\
&= \frac{c\cdot 10^{-16}}{h}+\frac{\cancel{\frac{\partial}{\partial\eta}f(\eta)}+\cancel{\nabla\left(\frac{\partial}{\partial\eta}f(\eta)\right)}\cdot h\cancel{\boldsymbol{d}}+(h\boldsymbol{d})^T Hf(\eta)(h\boldsymbol{d})-\cancel{\frac{\partial}{\partial\eta}f(\eta)}}{h}-\cancel{[Hf(\eta)d]} \\
&= \frac{c\cdot 10^{-16}}{h}+\frac{h}{h}+\frac{h^{\cancel{2}}\boldsymbol{d}^T Hf(\eta)\boldsymbol{d}}{\cancel{h}}
\end{aligned}$$

take derivative in h, set to 0

$$\begin{aligned}
0 &= -\frac{c\cdot 10^{-16}}{h^2}+Hf(\eta)\boldsymbol{d} \\
h &= \sqrt{\frac{c\cdot 10^{-16}}{Hf(\eta)\boldsymbol{d}}}=10^{-8}.
\end{aligned}$$

However, when it came to implementing the neural net I found 10^{-8}, 10^{-7}, 10^{-6} much to small. I ended up using ϵ's closer to 10^{-2} and 10^{-3}.

(d) As a warm-up, write a function that applies conjugate gradient to solve an $Ax = b$ problem given A and b. See Sauer section 2.6.3 for the conjugate gradient iteration. Construct a symmetric, positive definite matrix A and test your function. Show that your function finds the solution in n steps for an $n \times n$ matrix.

```r
CG<-function(A,b){
    x<-rep(0 ,dim(A)[1])
    d<- r<- b-(A%*%x)
    iter=0
    repeat{
        iter=iter+1
        Alpha<-as.numeric(  (t(r)%*%r)/(t(d)%*%A%*%d) )
        xnext<-x+Alpha*d
        rnext<-r-Alpha*A%*%d
        if( all(as.numeric( round(rnext,7) ) == 0) == T ){break}
        Beta<- as.numeric((t(rnext)%*%rnext)/(t(r)%*%r))
        dnext<-rnext+Beta*d
        d<-dnext;r<-rnext;x<-xnext
    }
    return(list(x=xnext,iterations=iter)) }

AA<-matrix(c(2,-1,0,-1,2,-1,0,-1,2),3,3,byrow=T)
```

```r
bb<-matrix(c(2,2,6),3,1)
solve(AA,bb)
```

```
##      [,1]
## [1,]    4
## [2,]    6
## [3,]    6
```

```r
eigen(AA)$values;dim(AA)
```

```
## [1] 3.4142136 2.0000000 0.5857864
```

```
## [1] 3 3
```

```r
CG(AA,bb)
```

```
## $x
##      [,1]
## [1,]    4
## [2,]    6
## [3,]    6
##
## $iterations
## [1] 3
```

It does well for a 3×3 matrix. We can try a somewhat larger one.

```r
dat<-mtcars[,c(1,3:6)]
dat<-var(dat)
eigen(dat)$values;dim(dat)
```

```
## [1] 18636.7869205  1452.9871142     9.2074199     0.1368769     0.1157957
```

```
## [1] 5 5
```

```r
bb<-rpois(dim(dat)[1],3)
solve(dat,bb)
```

```
##         mpg        disp          hp        drat          wt
##  1.60691186 -0.03236376  0.03385523 16.05679922 19.00571958
```

```r
CG(dat,bb)
```

```
## $x
##             [,1]
## mpg   1.60691186
## disp -0.03236376
## hp    0.03385523
## drat 16.05679922
## wt   19.00571958
##
## $iterations
## [1] 6
```

For this matrix it takes $n + 2$ iterations to get this one, but the stopping condition is somewhat large.

(e) Put together (a), (c) and (d) to apply Martens algorithm to train the neural net. You will need to modify your code from (d), so that the conjugate gradient uses the Hessian free estimate from (c). In the notation of (a), your conjugate gradient needs to solve the $Ax = b$ problem with $A = Hf(\eta) + \lambda I$ and $b = -\nabla f(\eta)$. You can use Martens method of backtracking using the conjugate gradient iterates or just set $\eta^{(i+1} = \eta^{(i)} + p$ where p is the solution of the "$Ax = b$" problem computed by conjugate gradient. Train your net and visualize the resultant classifiers.

Neural net functions.

```
NN<-function(x,eta,m){

    X<-as.matrix(x)

    Alpha<-matrix(eta[1:prod( c(dim(X)[2] + 1, m ) )], dim(X)[2] + 1 , m)
    Z<-apply(Alpha, 2, function(w) 1/(1+exp(- w[1] -   X %*% w[-1]  )) )

    Beta<-matrix( eta[(prod(dim(Alpha))+1):length(eta) ], dim(Z)[2]+1, 2)

    TT<-apply(Beta, 2, function(w) 1/(1+exp(- w[1] - Z %*% w[-1]  )) )
    Y<- apply(TT,2, function(x) exp(x))
    Y<-Y/rowSums(Y)
    return(Y)
}

logL<-function(x,y,eta,m){
    Y<-NN(x,eta,m)
    return(- sum( (1-y)*log(Y[,1]) + ( y)*log(Y[,2])  ) )   }

gradL<-function(x,y,eta,m ){

    y<-as.matrix(y)
    dimeta<-m*(dim(x)[2]+1)+2*(1+m)
    gradf<-rep(NA,dimeta)

    Y<-NN(x,eta,m)
    ETA<-eta

    logl<-logL(x,y,eta,m)
    for( i in 1:dimeta){
        ETA <-eta
        ETA[i]<-eta[i]+(10^-6)

        gradf[i]<- ( logL(x,y,ETA,m) -  logl  ) / (10^-6)   }
    return(gradf)
}
```

Train the neural net.

```
nn<-read.table("nn.txt",header=T)
set.seed(2221);theta<- runif(22,-.1,.1)
lambda =  100
```

```
#########       start       #########
for( k in 1:500){
  b<- (-gradL(nn[,-3],nn[,3],theta,4) )

  x<-rep(0,length(b))
  d<-r<- b    #Ax here is the zero vector, saves some computations
  plist<-list()

  for(i in 1:30){
    hfd<-   (Hfd(nn[,-3],nn[,3],theta,4,d,(10^-2))  + lambda * d)
    Alpha<-as.numeric(  (t(r)%*%r)/(t(d)%*% ( hfd   )) )
    xnext<-x+Alpha*d
    rnext<-r-Alpha*( hfd  )
    Beta<- as.numeric((t(rnext)%*%rnext)/(t(r)%*%r))
    dnext<-rnext+Beta*d
    d<-dnext;r<-rnext;x<-xnext
  }

  oldtheta=theta
  theta=xnext+theta

  if( logL(nn[,-3],nn[,3],theta,4) < logL(nn[,-3],nn[,3],oldtheta,4) ){
      lambda = lambda*9/10 } else { theta=oldtheta;lambda = 1110 }
  if( lambda < 2){  lambda =  4 }
}
```

Most of my success came when we did some controlling of λ. I started it off large, and whittled it down at each iteration if the log likelihood was lower, and generally it was. However, if λ went to low, it tended to blow up the log likelihood, so I kept λ between 2 and 4 when I was making progress. But if the log likelihood does increase during an iteration, I throw away the new theta, use the old theta and make λ 200. From there λ gets whittled down again. The log likelihood is at 622.6902

```
theta
```

```
##  [1]   -2.68665338    2.82821713    0.03636461   -2.10516779   -2.19735141
##  [6]   -0.23586784    8.04427640   -0.02610174   10.08045839   -8.57809443
## [11]   -0.10601871   11.60973171   13.79678982  220.53849949  238.01602030
## [16] -213.15047992  162.92599528  -13.80002292 -220.53812725 -238.01622445
## [21]  213.14654741 -162.92676571
```

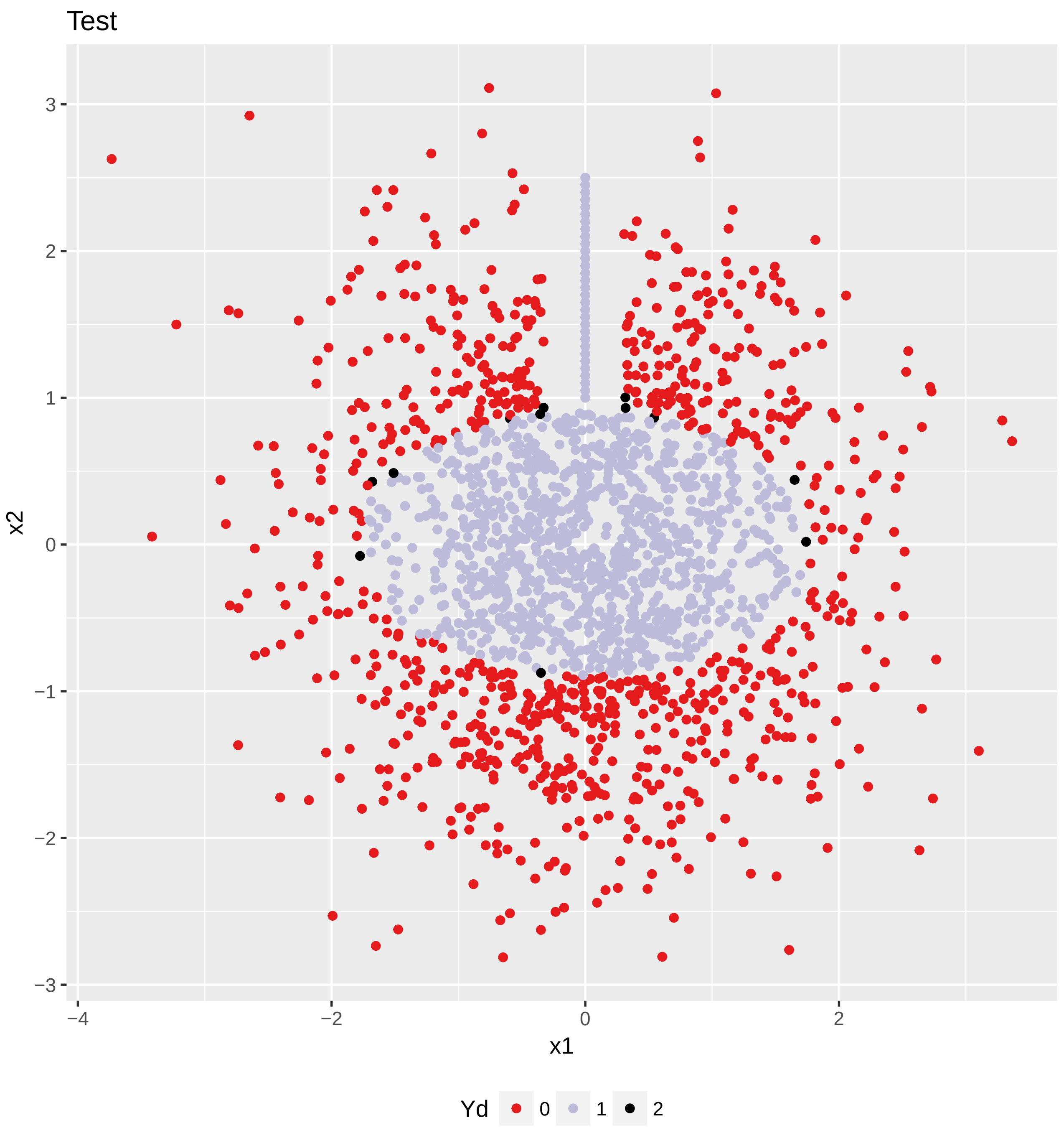
```
logL(nn[,-3],nn[,3], theta,4)
```

```
## [1] 622.6902
```

```
require(ggplot2, quietly = T)
test<-as.data.frame(( NN(nn[,-3],theta,4 )))
test<-cbind(test,nn[,-3]);test$y3<- ifelse( test[,1]>.5 , 0,1)
names(test)[3:4]<-c("x1","x2");test$Y<-nn$y;test$Yd<- (test$y3+test$Y)
test[which(test[,7] == 1),7]<-3;test[which(test[,7] == 2), 7]<-1
test[which(test[,7] == 3), 7]<-2;test$Yd<- as.factor(test$Yd)

cols <- c("#e41a1c","#bebada","black");names(cols ) <- levels(test$Yd)
colScale <- scale_colour_manual(name = "Yd",values = cols)
```

```
qq<- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(Yd))) +
          theme(legend.position="bottom")+ ggtitle("Test")      + colScale
```

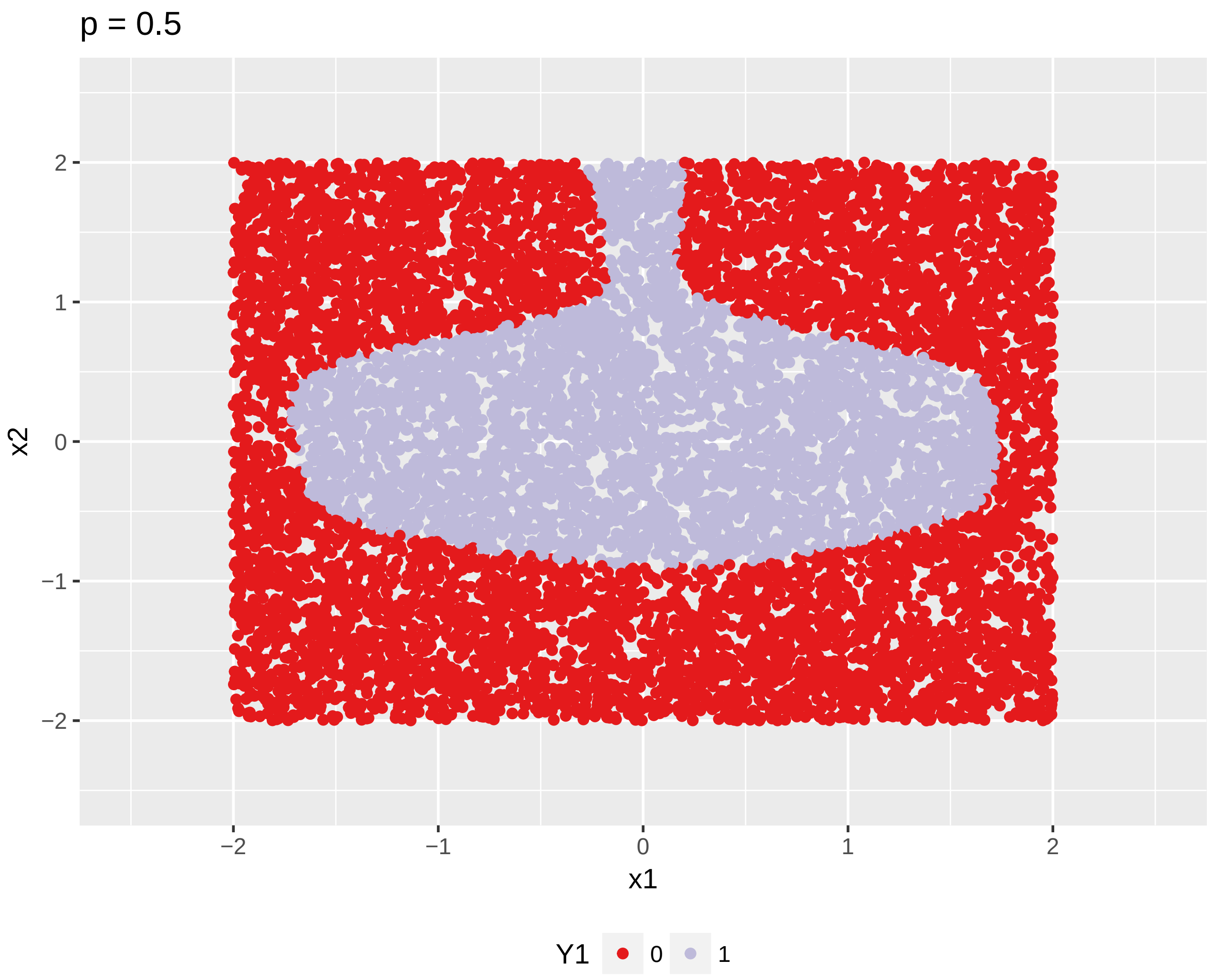
```
qq
```



The black dots are the ones the NN are assigning incorrectly; the red dots are correctly predicted class 0 and the lavender dots are correctly predicted class 1.

```
x1<-matrix(4*runif(20000 )-2, 10000,2);test<-as.data.frame((NN(x1,theta ,4)))
test<-cbind(test,x1);test$y1<- ifelse( test[,1]>.5 , 0,1)
test$y2<- ifelse( test[,1]>.7 , 0,1);test$y3<- ifelse( test[,1]>.3 , 0,1)
names(test)[3:4]<-c("x1","x2")
cols <- c("#e41a1c","#bebada");names(cols ) <- levels(test$Y1)
colScale <- scale_colour_manual(name = "Y1",values = cols)

p1 <- ggplot(test, aes(x1, x2)) + geom_point(aes(colour = factor(y1))) +
 ylim(-2.5, 2.5) + xlim( -2.5,2.5 ) + ggtitle("p = 0.5") +
 theme(legend.position="bottom") + colScale
```

```
p1
```


p = 0.5
x2
2
1
0
−1
−2
−2
−1
0
1
2
x1
Y1
0
1