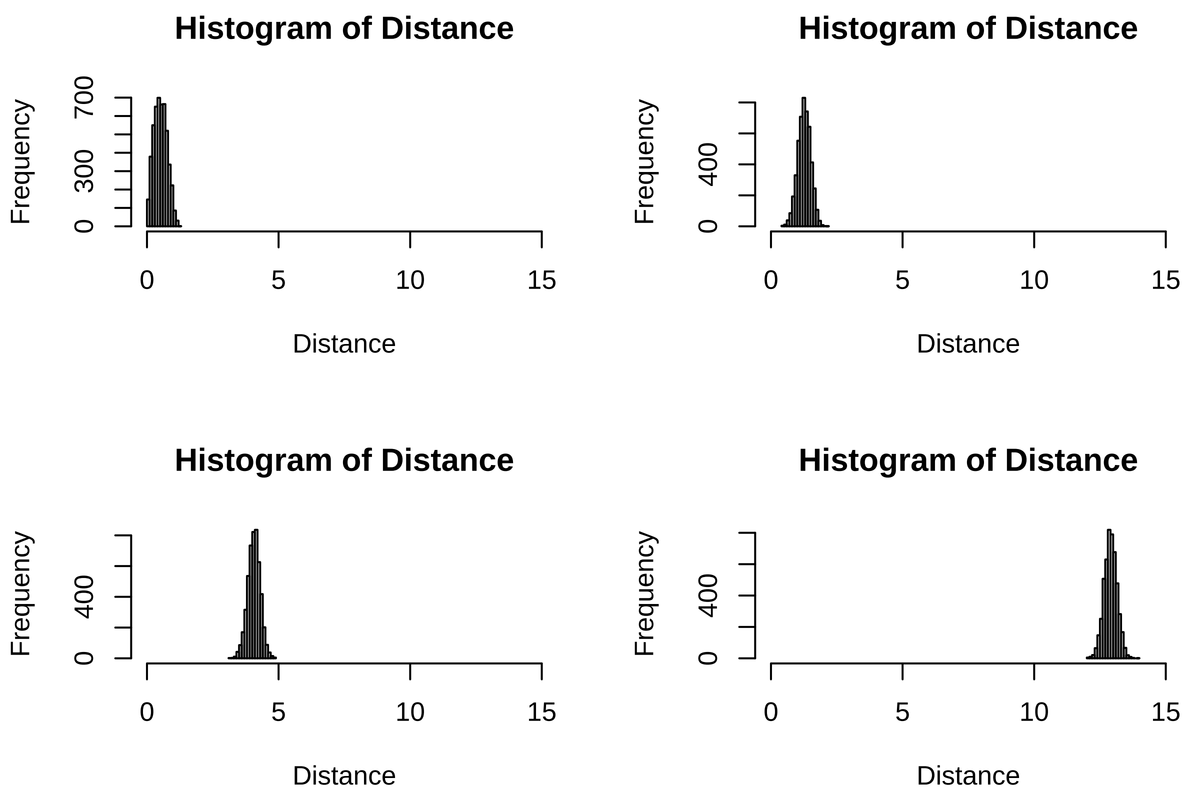
STAT602_HW1

1.6

As p becomes larger, the data points become sparse. Suppose we are going to use knn, then the nearest neighbors may not be close to the point that we want to make prediction. So it may not work well when we have high dimensions.

```
n <- 100
p <- c(2, 10, 100, 1000)
old<-par(no.readonly = TRUE)
par(mfrow = c(2,2))
for(i in 1:4){
  d <- p[i]
  data <- matrix(runif(d*n), nrow = n, ncol = d)
  dis <- numeric(choose(100,2))
  cursor <- 1
  for(j in 1:(n-1))
    for(k in 2:n){
      if(k>j){
        dis[cursor] <- sum((data[j,] - data[k,])^2)^(1/2)
        cursor = cursor + 1
      }
    }
  hist(dis, xlim = c(0, 15), xlab = "Distance", main = "Histogram of Distance")
}
```




```r
par(old)
```

1.7

We can see that the last two combinations are too large that it is out of bound.

```r
epsilon <- c(1,.1,.01)
p <- c(20, 50, 200)
```

```r
i=1;j=1
v = pi^(p[j]/2)/gamma(p[j]/2+1)*epsilon[i]^p[j]
(n = 1/v)
```

```
## [1] 38.74934
```

```r
i=2;j=1
v = pi^(p[j]/2)/gamma(p[j]/2+1)*epsilon[i]^p[j]
(n = 1/v)
```

```
## [1] 3.874934e+21
```

```r
i=3;j=1
v = pi^(p[j]/2)/gamma(p[j]/2+1)*epsilon[i]^p[j]
(n = 1/v)
```

```
## [1] 3.874934e+41
```

```r
i=1;j=2
v = pi^(p[j]/2)/gamma(p[j]/2+1)*epsilon[i]^p[j]
(n = 1/v)
```

```
## [1] 5.779614e+12
```

```r
i=2;j=2
v = pi^(p[j]/2)/gamma(p[j]/2+1)*epsilon[i]^p[j]
(n = 1/v)
```

```
## [1] 5.779614e+62
```

```r
i=3;j=2
v = pi^(p[j]/2)/gamma(p[j]/2+1)*epsilon[i]^p[j]
(n = 1/v)
```

```
## [1] 5.779614e+112
```

```r
i=1;j=3
v = pi^(p[j]/2)/gamma(p[j]/2+1)*epsilon[i]^p[j]
(n = 1/v)
```

```
## [1] 1.798939e+108
```

```r
i=2;j=3
v = pi^(p[j]/2)/gamma(p[j]/2+1)*epsilon[i]^p[j]
(n = 1/v)
```

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

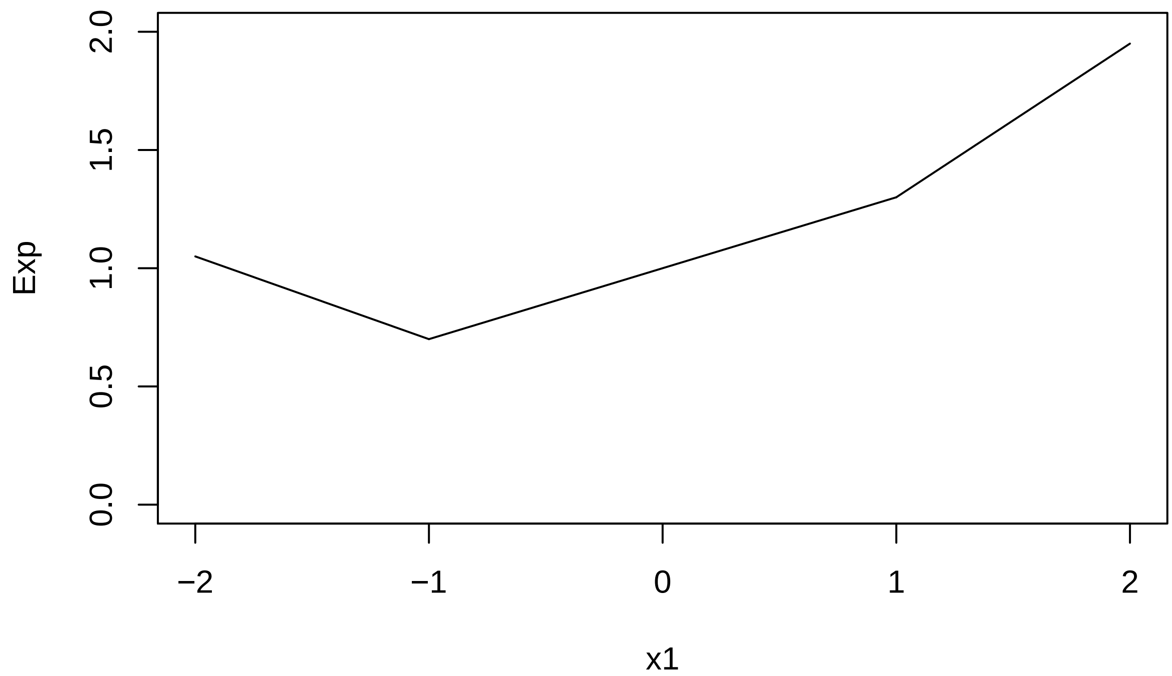
```r
i=3;j=3
v = pi^(p[j]/2)/gamma(p[j]/2+1)*epsilon[i]^p[j]
(n = 1/v)
```

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

2.4

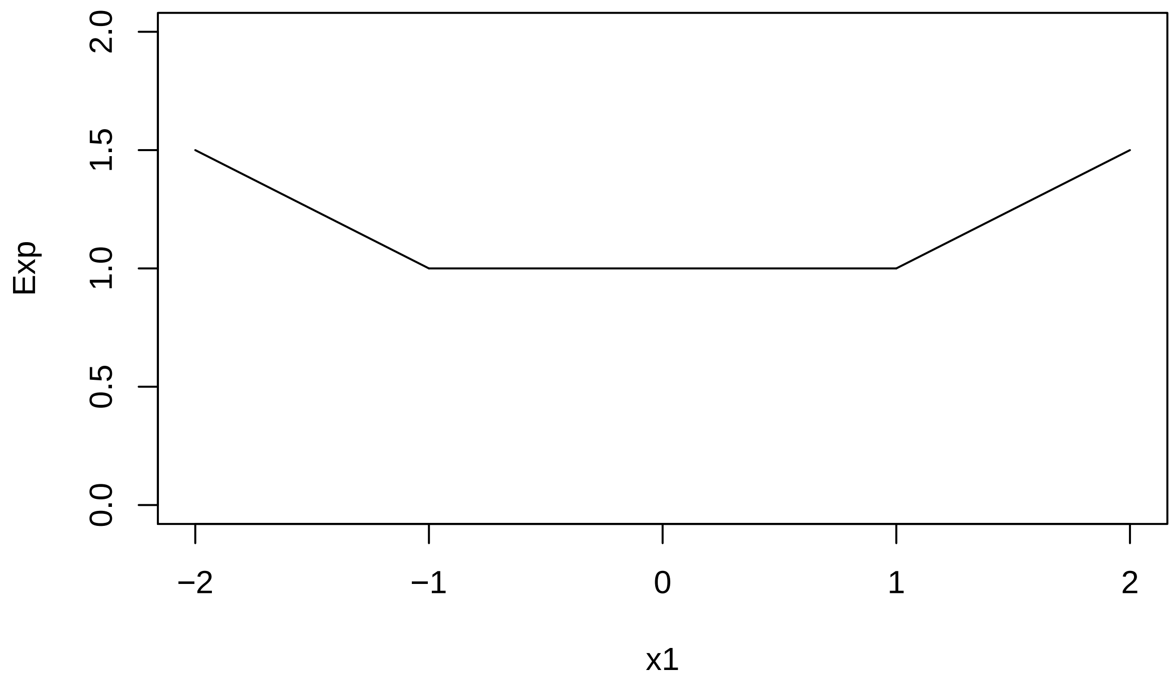
```r
p <- c(0.35, 0.65)
E <- function(p, y){
  value <- p*max(0,1-y)+(1-p)*max(1+y,0)
  return(value)
}
x1 <- seq(-1,1,by = 0.01)
Exp <- numeric(length(x1))
for(i in 1:length(x1)){
  Exp[i] <- E(p[1], x1[i])
}
plot(x1, Exp, ty = "l", xlim = c(-2,2), ylim = c(0,2))

x1 <- seq(1,2,by = 0.01)
Exp <- numeric(length(x1))
for(i in 1:length(x1)){
  Exp[i] <- E(p[1], x1[i])
}
lines(x1, Exp)
```

```r
x1 <- seq(-2,-1,by = 0.01)
Exp <- numeric(length(x1))
for(i in 1:length(x1)){
  Exp[i] <- E(p[1], x1[i])
}
lines(x1, Exp)
```




```r
p <- c(0.5,0.35)
E <- function(p, y){
  value <- p*max(0,1-y)+(1-p)*max(1+y,0)
  return(value)
}
x1 <- seq(-1,1,by = 0.01)
Exp <- numeric(length(x1))
for(i in 1:length(x1)){
  Exp[i] <- E(p[1], x1[i])
}
plot(x1, Exp, ty = "l", xlim = c(-2,2), ylim = c(0,2))

x1 <- seq(1,2,by = 0.01)
Exp <- numeric(length(x1))
for(i in 1:length(x1)){
  Exp[i] <- E(p[1], x1[i])
}
lines(x1, Exp)
```

```
x1 <- seq(-2,-1,by = 0.01)
Exp <- numeric(length(x1))
for(i in 1:length(x1)){
  Exp[i] <- E(p[1], x1[i])
}
lines(x1, Exp)
```



3.4

$\{1, sin(x), cos(x)\}$ is the best. If the data is generated by factors that is not included in the last model, then none of these model is without model bias. If the data is generated with one of these model, then including that one, the models that include the true model are without the model bias.

```
data <- read_excel("~/Downloads/Problem3.4Data.xlsx")

K = 10
SSE  = numeric(9)
for(i in 1:K)
{
  test = data[(10*(i-1)+1):(10*i),]
  train = data[-((10*(i-1)+1):(10*i)),]

  # Polynomial
  ## 0
  fp0 <- lm(data = train, y~1)
```

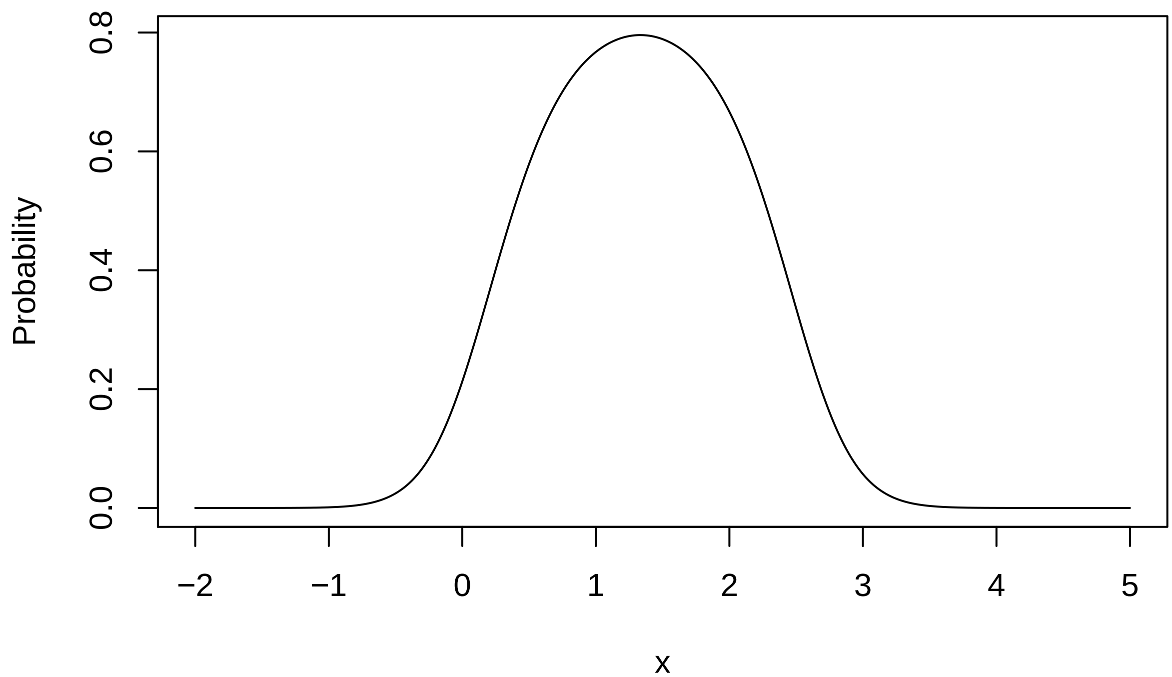
```
  yp0 <- predict(fp0, test)
  SSE[1] = SSE[1] + sum((yp0-test$y)^2)

  ## 1
  fp1 <- lm(data = train, y~x)
  yp1 <- fp1$coefficients %*% t(as.matrix(data.frame(inter = 1, x = test$x)))
  SSE[2] = SSE[2] + sum((yp1-test$y)^2)

  ## 2
  fp2 <- lm(data = train, y~x+I(x^2))
  yp2 <-  fp2$coefficients %*% t(data.frame(inter = 1, x = test$x, x2 = test$x^2))
  SSE[3] = SSE[3] + sum((yp2-test$y)^2)

  ## 3
  fp3 <- lm(data = train, y~x+I(x^2)+I(x^3))
  yp3 <- fp3$coefficients %*% t(data.frame(inter = 1,x = test$x, x2 = test$x^2, x3 = test$x^3))
  SSE[4] = SSE[4] + sum((yp3-test$y)^2)

  ## 4
  fp4 <- lm(data = train, y~x+I(x^2)+I(x^3)+I(x^4))
  yp4 <- fp4$coefficients %*% t(data.frame(inter = 1, x = test$x, x2 = test$x^2, x3 = test$x^3, x4 = tes
  SSE[5] = SSE[5] + sum((yp4-test$y)^2)

  ## 5
  fp5 <- lm(data = train, y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5))
  yp5 <- fp5$coefficients %*% t(data.frame(inter =1,x = test$x, x2 = test$x^2, x3 = test$x^3, x4 = test$
  SSE[6] = SSE[6] + sum((yp5-test$y)^2)

  ## 6
  fp6 <- lm(data = train, y~1+I(sin(x))+I(cos(x)))
  yp6 <- fp6$coefficients %*% t(as.matrix(data.frame(inter = 1,sinx = sin(test$x), cosx = cos(test$x)))
  SSE[7] = SSE[7] + sum((yp6-test$y)^2)

  ## 7
  fp7 <- lm(data = train, y~1+I(sin(x))+I(cos(x))+I(sin(2*x))+I(cos(2*x)))
  yp7 <- fp7$coefficients %*% t(as.matrix(data.frame(inter = 1,sinx = sin(test$x), cosx = cos(test$x), s
  SSE[8] = SSE[8] + sum((yp7-test$y)^2)

  ## 8
  fp8 <- lm(data = train, y~1+x+I(x^2)+I(x^3)+I(x^4)+I(x^5)+
              I(sin(x))+I(cos(x))+I(sin(2*x))+I(cos(2*x)))
  yp8 <- fp8$coefficients %*% t(data.frame(inter = 1 ,x = test$x, x2 = test$x^2,
                                 x3 = test$x^3, x4 = test$x^4,
                                 x5 = test$x^5 ,sinx = sin(test$x),
                                 cosx = cos(test$x), sin2x = sin(2*test$x),
                                 cos2x = cos(2*test$x)))
  SSE[9] = SSE[9] +sum((yp8-test$y)^2)
}
SSE
```

```
## [1] 207.8989 212.4108 206.4061 190.6987 187.3633 193.0590 178.7845 184.4818
## [9] 202.8212
```

3.11

(a)

```r
py1x <- function(x)
{
  val = dnorm(x,1,0.5)/(dnorm(x,1,0.5)+dnorm(x,0,1))
  return(val)
}

x <- seq(-2,5,by = 0.01)
plot(x,py1x(x),type = "l", ylab = "Probability")
```



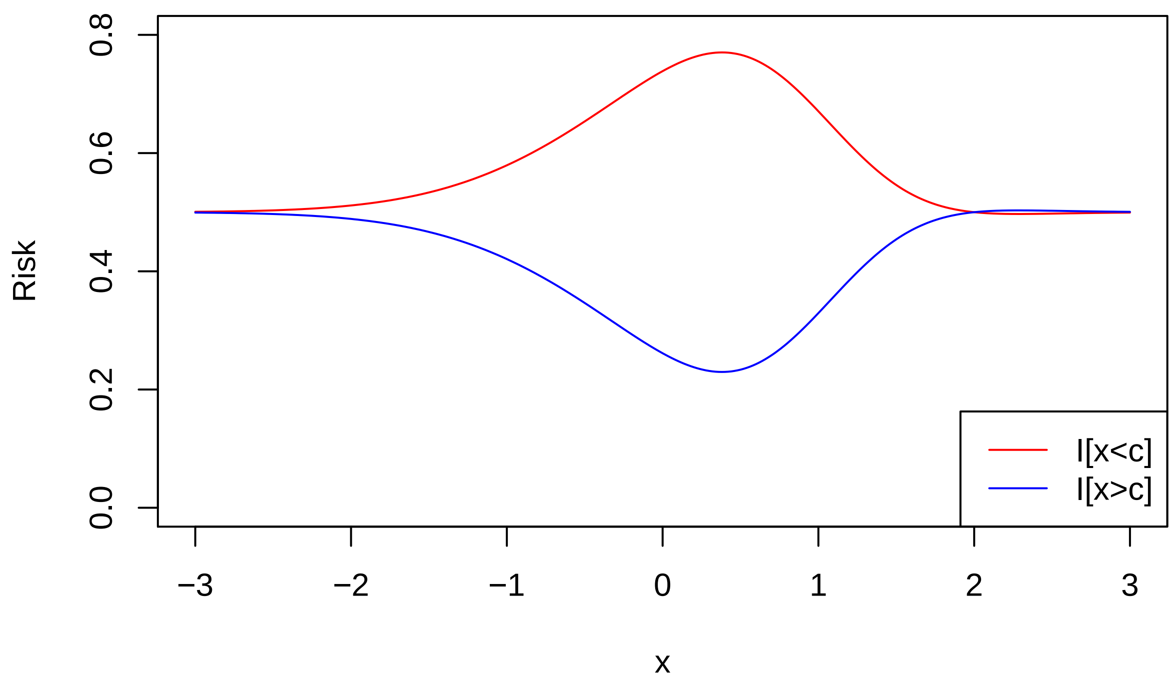
(b)

```r
x1 = sqrt(2/3*(2/3+log(2)))+4/3
x2 = -sqrt(2/3*(2/3+log(2)))+4/3

(EER <- 1/2*(pnorm(x1, 0, 1)-pnorm(x2, 0, 1))+
  1/2*(pnorm(x1, 1, 0.5,lower.tail = 0)+pnorm(x2, 1, 0.5)))
```

```
## [1] 0.2266941
```

(c)

```
linear1 <- function(c){
  value = (pnorm(c, 0, 1) + pnorm(c, 1, 0.5, lower.tail = 0))/2
  return(value)
}

linear2 <- function(c){
  value = (pnorm(c, 1, 0.5) + pnorm(c, 0, 1, lower.tail = 0))/2
  return(value)
}

x = seq(from = -3, to = 3, by = 0.02)
plot(x, linear1(x), ylim = c(0,0.8), type = "l", col = "red", ylab = "Risk")
legend("bottomright", c("I[x<c]", "I[x>c]"), col = c("red","blue"), lty = c(1,1))
lines(x, linear2(x), col = "blue")
```



By minimizing the risk.

```
op <- optim(0.1, linear2, method = "Brent", lower = 0, upper = 1)
op$par
```

```
## [1] 0.381208
```

So $g^* = I[x > 0.381]$

The error rate for g^* is:

```
op$value
```

```
## [1] 0.2297298
```

So the modeling penalty is:

```
op$value - EER
```

```
## [1] 0.003035748
```

(d)

```
M <- 10000
N <- 100

mini <- function(c)
{
  value = min(sum(Y[X<c]==0)+sum(Y[X>c]==1), sum(Y[X<c]==1)+sum(Y[X>c]==0))
  return(value)
}

Error = 0
for(i in 1:M){
  Y <- sample(0:1, N, replace = TRUE)
  X <- rnorm(N, Y, 1-0.5*Y)

  c = optim(0, mini, method = "Brent", lower = -3, upper = 3)$par

  if(sum(Y[X<c]==0)+sum(Y[X>c]==1) <= sum(Y[X<c]==1)+sum(Y[X>c]==0)){
    Error = Error + linear1(c)
  }
  else{
    Error = Error + linear2(c)
  }
}
Error = Error/M
```

When $N = 100$, the error rate is:

```
Error
```

```
## [1] 0.2411257
```

The "fitting penalty" is:

```
Error - op$value
```

```
## [1] 0.01139588
```

```
M <- 10000
N <- 50

mini <- function(c)
{
  value = min(sum(Y[X<c]==0)+sum(Y[X>c]==1), sum(Y[X<c]==1)+sum(Y[X>c]==0))
  return(value)
```

```
}

Error = 0
for(i in 1:M){
  Y <- sample(0:1, N, replace = TRUE)
  X <- rnorm(N, Y, 1-0.5*Y)

  c = optim(0, mini, method = "Brent", lower = -3, upper = 3)$par

  if(sum(Y[X<c]==0)+sum(Y[X>c]==1) <= sum(Y[X<c]==1)+sum(Y[X>c]==0)){
    Error = Error + linear1(c)
  }
  else{
    Error = Error + linear2(c)
  }
}
Error = Error/M
```

The error rate and the "fitting penalty" are:

```
Error;Error-op$value
```

```
## [1] 0.2499291
```

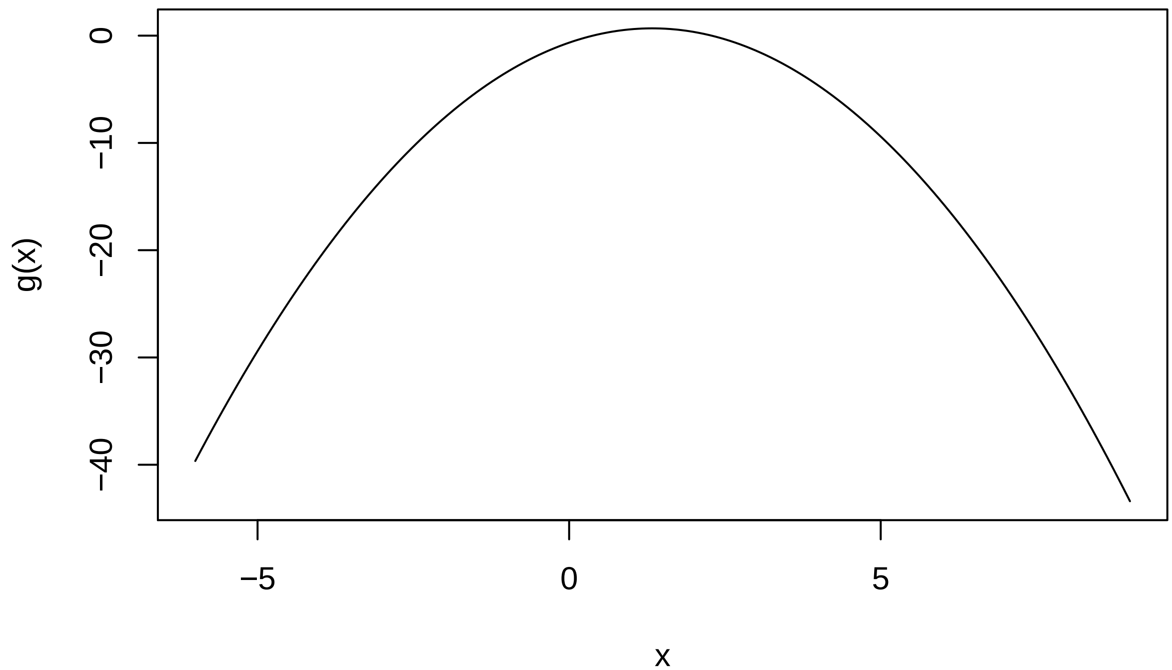
```
## [1] 0.02019927
```

And, indeed, when we have a smaller sample size, the fitting penalty is greater.

3.12

(a)

```
g <- function(x){
  value = 1/2*log(dnorm(x, 1, 0.5)/dnorm(x, 0, 1))
  return(value)
}
x <- seq(-6,9,by=0.02)
plot(x,g(x),ty = "l")
```



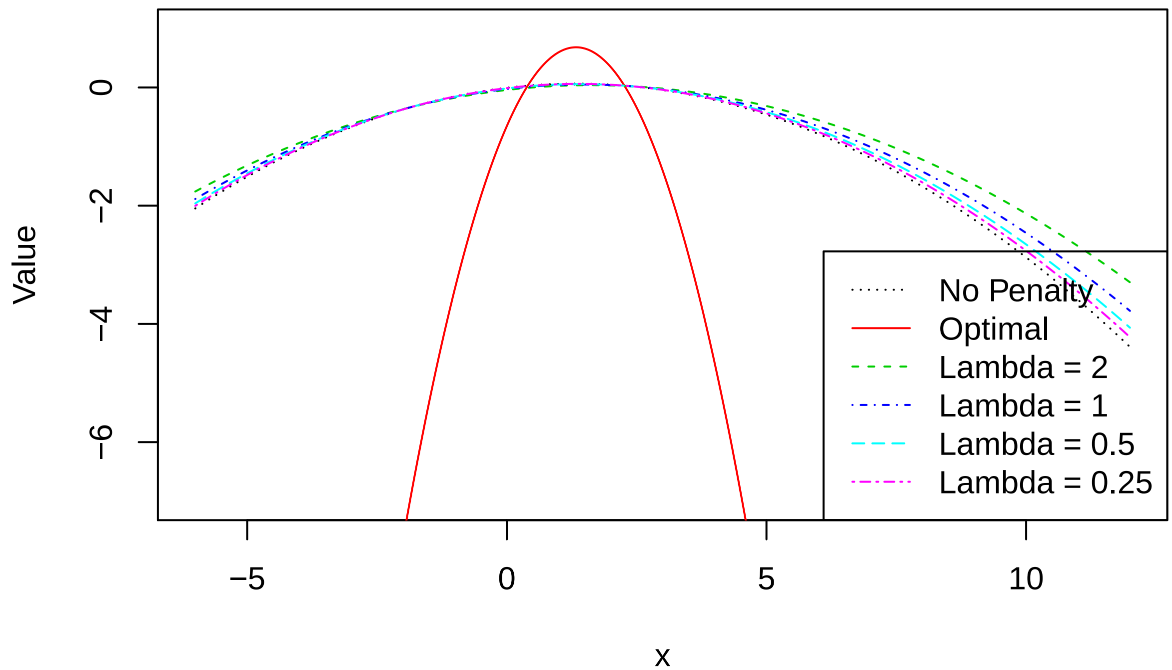
(b)

```
data2 <- read_excel("~/Downloads/Problem3.4Data.xlsx")
xx <- data2$x; y <- data2$y

func2 <- function(para)
{
  beta0 = para[1]
  beta1 = para[2]
  beta2 = para[3]

  value = mean(exp(-y*(beta0+beta1*(xx-mean(xx))+beta2*(xx-mean(xx))^2)))

  return(value)
}

op1 <- optim(c(0,0,0),func2)
op1$par
```

```
## [1]  0.03694171  0.06596512 -0.03919403
```

(c)

```
func_2 <- function(x, op){
  value = op$par[1]+op$par[2]*(x-mean(xx))+op$par[3]*(x-mean(xx))^2
  return(value)
}

x <- seq(-6,12,by=0.02)
plot(x, func_2(x, op1), ty = "l", ylab = "Value", lty = 3, col = 1, ylim = c(-7,1))
lines(x, g(x), lty = 1, col = 2)

data2 <- read_excel("~/Downloads/Problem3.4Data.xlsx")
xx <- data2$x; y <- data2$y

func3 <- function(para){
  beta0 = para[1]
  beta1 = para[2]
  beta2 = para[3]

  value = mean(exp(-y*(beta0+beta1*(xx-mean(xx))+beta2*(xx-mean(xx))^2))) + lambda*beta2^2

  return(value)
}

lambda = 2
op5 <- optim(c(0,0,0), func3)
lines(x, func_2(x, op5), lty = 2, col = 3)

lambda = 1
op4 <- optim(c(0,0,0), func3)
lines(x, func_2(x, op4), lty = 4, col = 4)

lambda = 0.5
op3 <- optim(c(0,0,0), func3)
lines(x, func_2(x, op3), lty = 5, col = 5)

lambda = 0.25
op2 <- optim(c(0,0,0), func3)
lines(x, func_2(x, op2), lty = 6, col = 6)

legend("bottomright",lty = c(3,1,2,4,5,6), col = 1:6,legend = c("No Penalty", "Optimal", "Lambda = 2",
                                                 "Lambda = 1", "Lambda = 0.5", "Lambda = 0.25"))
```


Value
0
−2
−4
−6
−5
0
5
10
x
No Penalty
Optimal
Lambda = 2
Lambda = 1
Lambda = 0.5
Lambda = 0.25