# Stat 207 HW6

Cheng Luo 912466499
Fan Wu 912538518

February 25, 2015

# 1 14.9

(a)

```r
dat = read.table("CH14PR09.txt")
names(dat) = c("Y", "X")
dat = dat[order(dat$X), ]
logit = glm(Y ~ X, data = dat, family = "binomial")
summary(logit)
```

```
##
## Call:
## glm(formula = Y ~ X, family = "binomial", data = dat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7845  -0.8350   0.5065   0.8371   1.7145
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.308925   4.376997  -2.355   0.0185 *
## X             0.018920   0.007877   2.402   0.0163 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 37.393  on 26  degrees of freedom
## Residual deviance: 29.242  on 25  degrees of freedom
## AIC: 33.242
##
## Number of Fisher Scoring iterations: 4
```

```r
b0 = coef(logit)[1]; b0
```

```
## (Intercept)
##   -10.30893
```

```r
b1 = coef(logit)[2]; b1
```

```
##          X
## 0.01891983
```

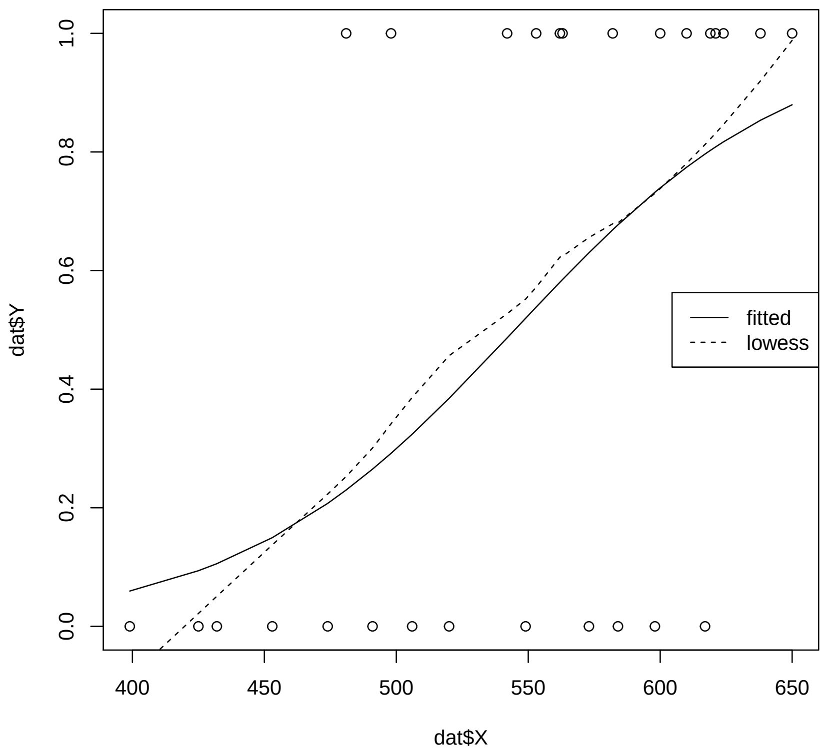
From the summary, the maximum likelihood estimates of $b_0 = -10.308925$,

$b_1 = 0.018920$,

$$\hat{\pi} = \frac{exp(b_0 + b_1 X)}{1 + exp(b_0 + b_1 X)} = \frac{exp(-10.308925 + 0.018920X)}{1 + exp(-10.308925 + 0.018920X)}$$

(b)

```
plot(dat$X, dat$Y)
points(dat$X, fitted(logit), type = 'l', lty = 1)
points(dat$X, lowess(dat$X, dat$Y)$y, type = 'l', lty = 2)
legend('right', legend = c('fitted', 'lowess'),
       lty = 1:2)
```




The fitted logistic response function appears to be well.

(c)

```
exp(0.018920)
```

```
## [1] 1.0191
```

$exp(\beta_1) = 1.0191$, so that the odds of employee's ability increased by 1.91% with each additional employee's emotional stability.

(d)

```
newdat = data.frame(X = 550)
predict(logit, newdata = newdat, type = "response")

##         1
## 0.5242263
```

The estimated probability that employees with an emotional stability test score of 550 will be able to perform in a task group is 0.5242263 .

# 2 Problem 5

(a)

```
dat = read.table("apartment.txt", header = TRUE)
require("pls")

## Loading required package:  pls

## Warning:  package 'pls' was built under R version 3.1.2

##
## Attaching package:  'pls'
##
## The following object is masked from 'package:stats':
##
##    loadings

dat.stan = dat
for(j in 1:ncol(dat))
  dat.stan[,j] = (dat[,j] - mean(dat[,j]))/sd(dat[,j])
n = nrow(dat); n

## [1] 25

fit = plsr(Y ~0 + ., data = dat.stan, 5, validation = "CV")
summary(fit)

## Data:  X dimension: 25 5
##  Y dimension: 25 1
## Fit method: kernelpls
## Number of components considered: 5
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
```

```
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps
## CV           1.021   0.3245   0.2416   0.1843   0.1718   0.2124
## adjCV        1.021   0.3186   0.2405   0.1802   0.1692   0.2082
##
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps
## X    52.58    66.94    85.17    91.35   100.00
## Y    92.14    96.53    97.92    98.01    98.05
```

```
scores(fit)
```

```
##         Comp 1      Comp 2      Comp 3       Comp 4       Comp 5
## 1  -1.43809414  0.31257891 -0.66465691 -0.238439173  0.159569798
## 2   1.65010464 -1.22067017 -0.64427193  0.663274381 -0.865798881
## 3  -1.05278777  0.23602481 -0.31050170 -0.020291335  0.008296669
## 4   2.04913072  1.58847399  1.43430866  0.056609088  0.998283179
## 5  -1.13936477  0.29816779  0.09920242 -0.077974453 -0.315614625
## 6   0.43417694 -0.26004572 -1.00952515  0.650608213  0.643434804
## 7  -1.22623584 -0.17772037 -1.20001375 -0.068573074  0.280347608
## 8  -0.65767426  0.36330099  1.03967609 -0.268877399 -1.370326788
## 9   1.42140486 -1.39664727  1.71050459 -0.349167349  1.208806076
## 10  5.48855488  0.07922404 -1.42064358 -0.711330743 -0.865767632
## 11  1.98211289  1.07654279 -0.19360273  0.247458345  0.426744426
## 12  0.04614307 -0.64535432  0.61846253 -0.480354282  1.057377814
## 13 -0.71466027  0.75759510 -0.14154088 -0.168335371  0.116132220
## 14 -0.89066011 -0.24878365  1.32675710 -0.194652611 -0.421211010
## 15  0.41103483 -1.04629327  0.48064460  0.246649352  0.306896982
## 16 -1.02148397 -0.21942028  1.30789234 -0.154167563 -0.564857509
## 17  0.77416093 -0.20479803 -0.70800362 -0.161374329 -0.174850186
## 18 -1.24116621  0.12113528 -0.52207912 -0.071041829  0.036778165
## 19 -0.36723658 -0.32047183  0.43733622  0.267294081 -0.910453651
## 20 -1.18318143  0.27366081  0.02535909 -0.014259103 -0.184750720
## 21 -1.21222218 -0.07162508 -1.09128123 -0.058437672  0.309847588
## 22 -1.43423671  0.17603426  0.16663071 -0.203393131 -0.594314019
## 23  1.20087805  0.46813038  1.30158137  0.899514178 -0.313846232
## 24 -1.14387561 -0.16195504 -1.19339577 -0.004177028  0.359346058
## 25 -0.73482194  0.22291589 -0.84883933  0.213438810  0.669929866
## attr(,"class")
## [1] "scores"
## attr(,"explvar")
##    Comp 1    Comp 2    Comp 3    Comp 4    Comp 5
## 52.582242 14.361573 18.228133  6.180191  8.647861
```

```
loadings(fit)[, 1:3]
```

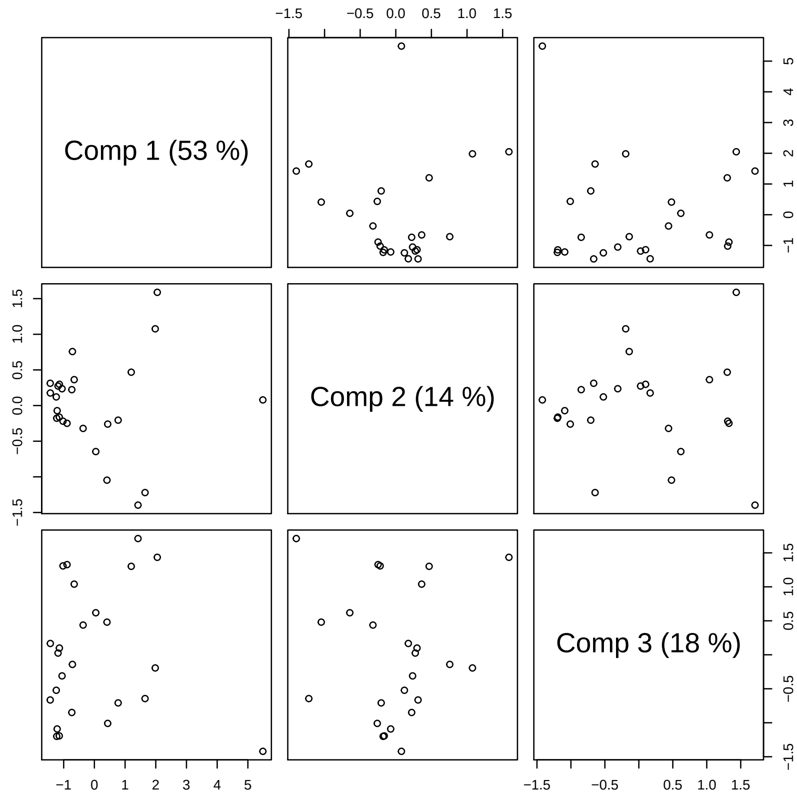
```
##         Comp 1     Comp 2      Comp 3
## X1 -0.07983743  0.6903977 -0.76418606
## X2  0.59805736  0.1008482 -0.08164449
## X3  0.54164949 -0.5440785 -0.25282151
## X4  0.16890957 -0.7416964  0.59414237
## X5  0.56738864  0.5744813  0.04300711
```

```
k = 1:6
r.sq = c(92.14 ,   96.53  ,  97.92  ,  98.01 ,   98.05)/100
r.adj = 1 - (n-1)/(n-k-1)*(1-r.sq); r.adj
```

```
## Warning in (n - 1)/(n - k - 1) * (1 - r.sq):  longer object length
is not a multiple of shorter object length
```

```
## [1] 0.9179826 0.9621455 0.9762286 0.9761200 0.9753684 0.8952000
```

```
plot(fit, plottype = "scores", comps = 1:3)
```




The first component is the most important component to predict the data and each component seems to be uncorrelated with other component.

(b)

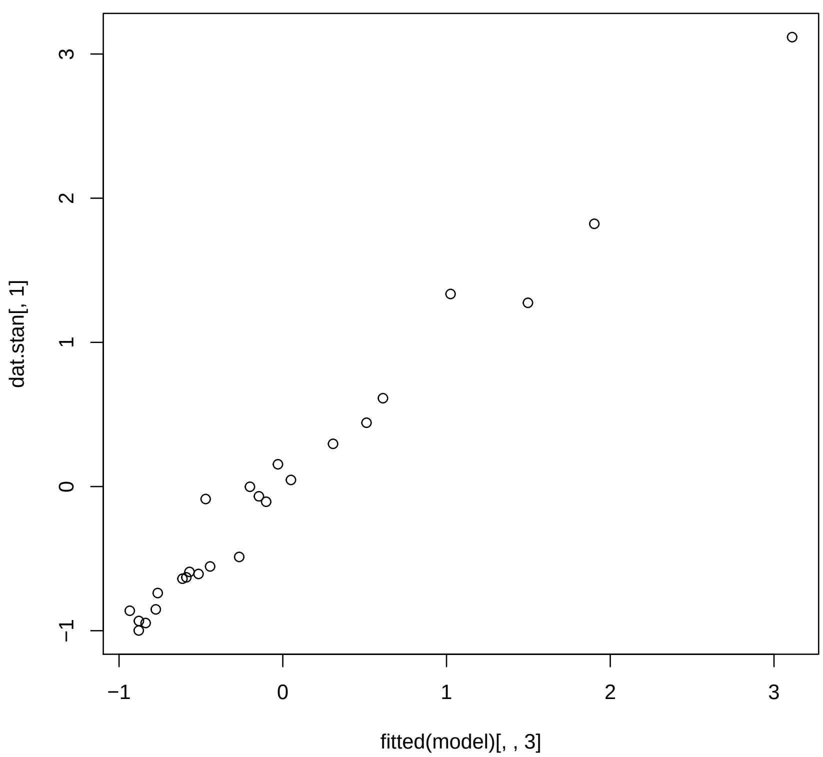
```
r.sq = c(0, r.sq); r.sq

## [1] 0.0000 0.9214 0.9653 0.9792 0.9801 0.9805

f.k = sapply(2:length(r.sq), function(k)
  (n-k-1)*(r.sq[k] - r.sq[k-1])/(1-r.sq[k])); f.k

## [1] 257.8982188  26.5677233  13.3653846   0.8592965   0.3692308

qf(1-0.05, 1, 1:5)

## [1] 161.447639  18.512821  10.127964   7.708647   6.607891
```

As we can see from above, we might decide the number of components to keep is 3.

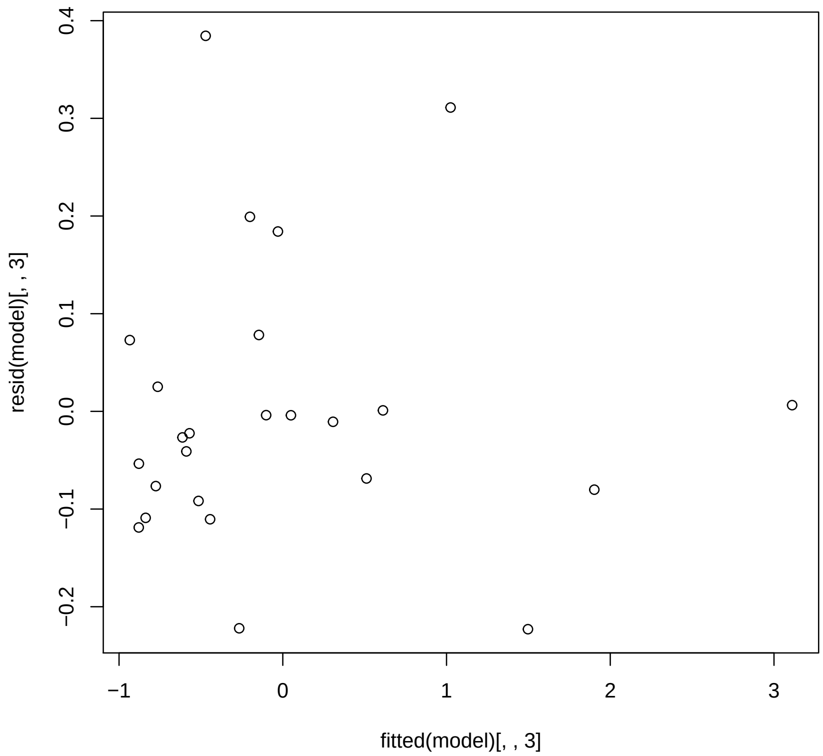
(c)
```r
model = plsr(Y ~ 0 + .,
             3, data = dat.stan, validation = 'CV')
coef(model)
```

```
## , , 3 comps
##
##               Y
## X1 -0.11356364
## X2  0.34543343
## X3 -0.02384503
## X4  0.05143543
## X5  0.67482746
```

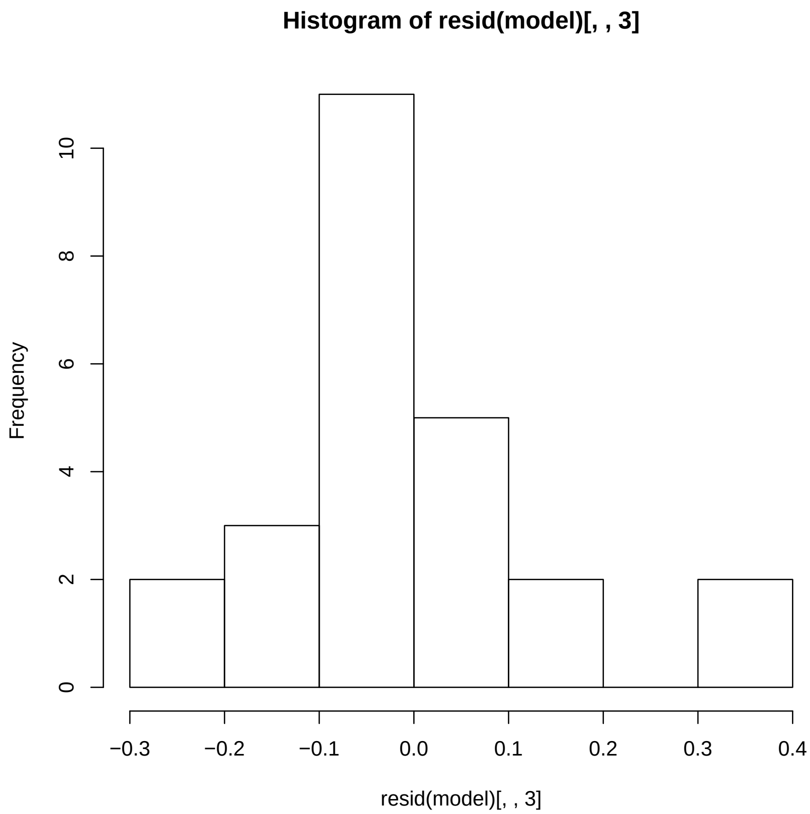
```r
plot(fitted(model)[,,3], dat.stan[,1])
```



```r
plot(fitted(model)[,,3], resid(model)[,,3])
```




```r
hist(resid(model)[,,3])
```




The final model is

$$Y^* = -0.11356364X_1^* + 0.34543343X_2^* - 0.02384503X_3^* + 0.05143543X_4^* + 0.67482746X_5^*$$

The observed against the fitted values plots shows it fits well, and residuals against the fitted values and the histogram of the residuals plots show it has no sign for unequal variance.

# 3 Problem 6

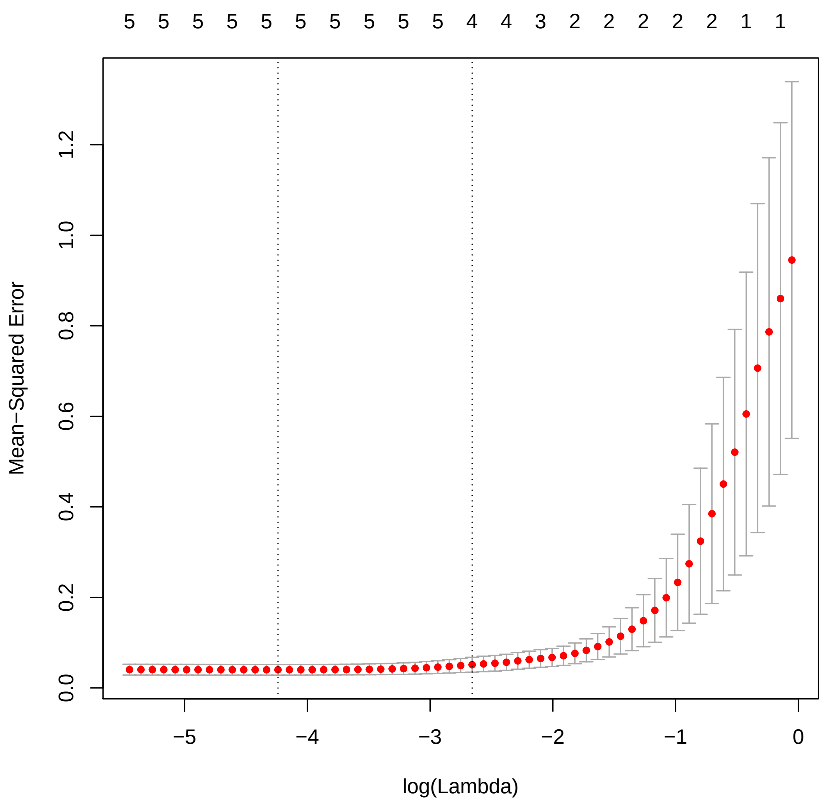
(a)

```
require(glmnet)
## Loading required package:  glmnet
## Warning:  package 'glmnet' was built under R version 3.1.2
## Loading required package:  Matrix
## Warning:  package 'Matrix' was built under R version 3.1.2
## Loaded glmnet 1.9-8
```

```r
x = as.matrix(dat.stan[, -1])
model = cv.glmnet(x, dat.stan[, 1], intercept = FALSE)
```

```
## Warning:  Option grouped=FALSE enforced in cv.glmnet, since <
3 observations per fold
```

```r
plot(model)
```




```r
model$lambda.min
```

```
## [1] 0.01441744
```

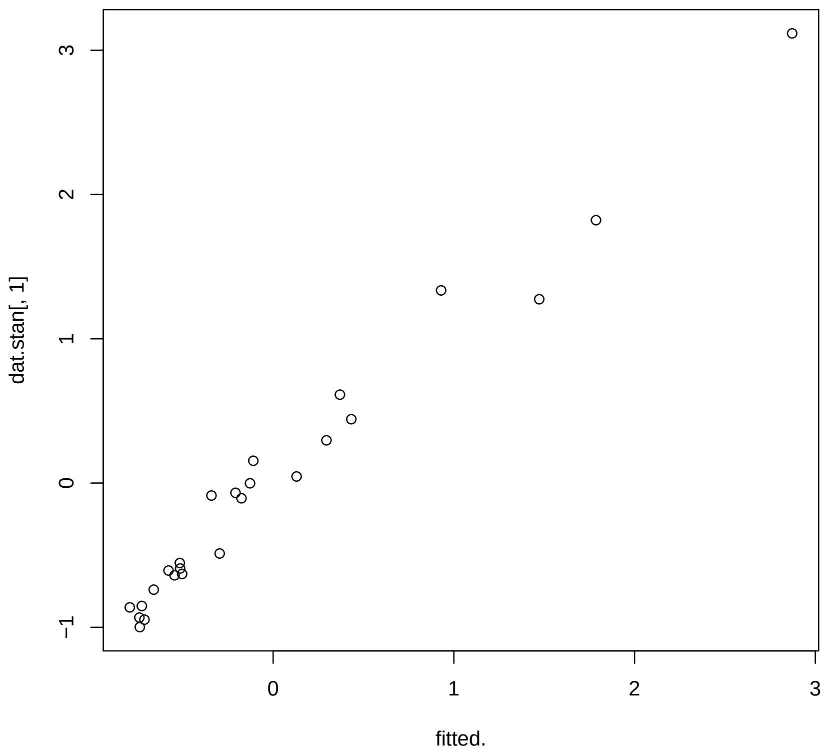
(b)

```r
coef(model)
```

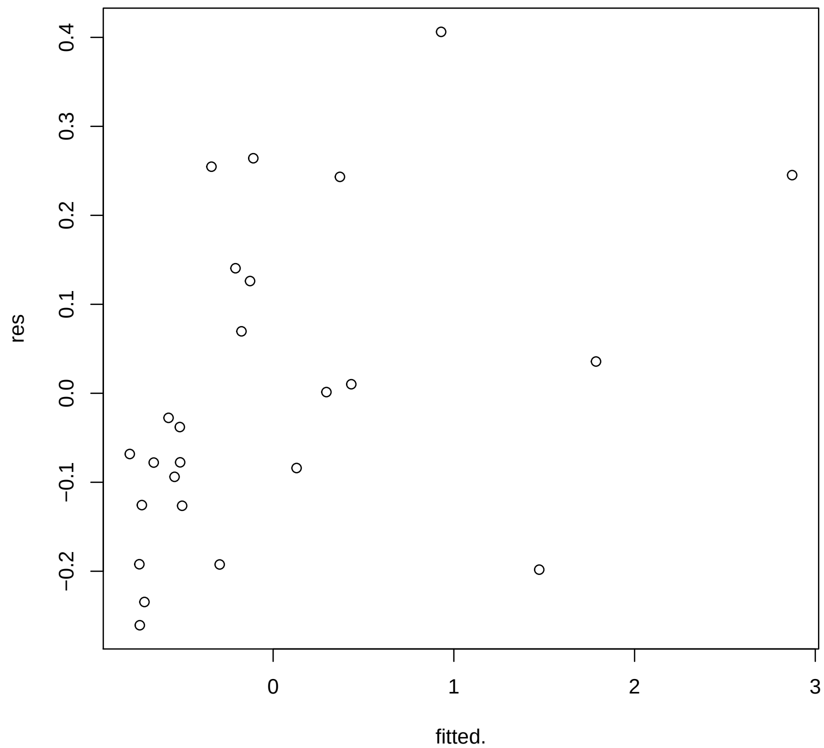
```
## 6 x 1 sparse Matrix of class "dgCMatrix"
```

```
##                         1
## (Intercept)  .
## X1          -0.05131094
## X2           0.26469804
## X3           .
## X4           0.01620098
## X5           0.66414967
```

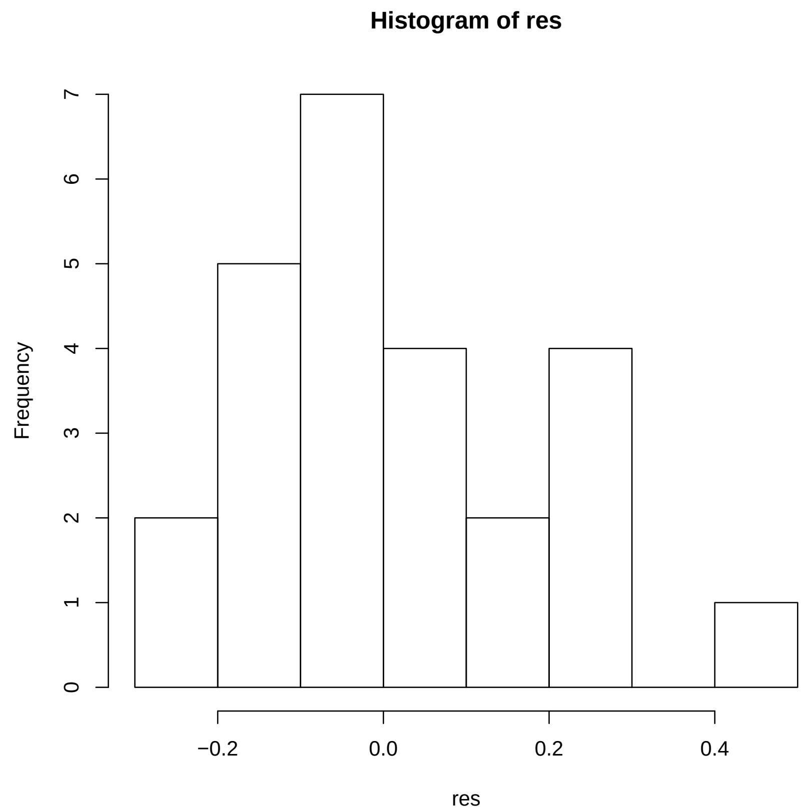
```r
fitted. = predict(model, newx = x)
plot(fitted., dat.stan[,1])
```




```r
res = dat.stan[,1] - fitted.
plot(fitted., res)
```




```r
hist(res)
```




The final model is

$$Y^* = -0.0610260558X_1^* + 0.2646220373X_2^* + 0.0002055733X_3^* + 0.0237525845X_4^* + 0.6758144083X_5^*$$

The observed against the fitted values plots shows it fits well, and residuals against the fitted values and the histogram of the residuals plots show it has no sign for unequal variance.

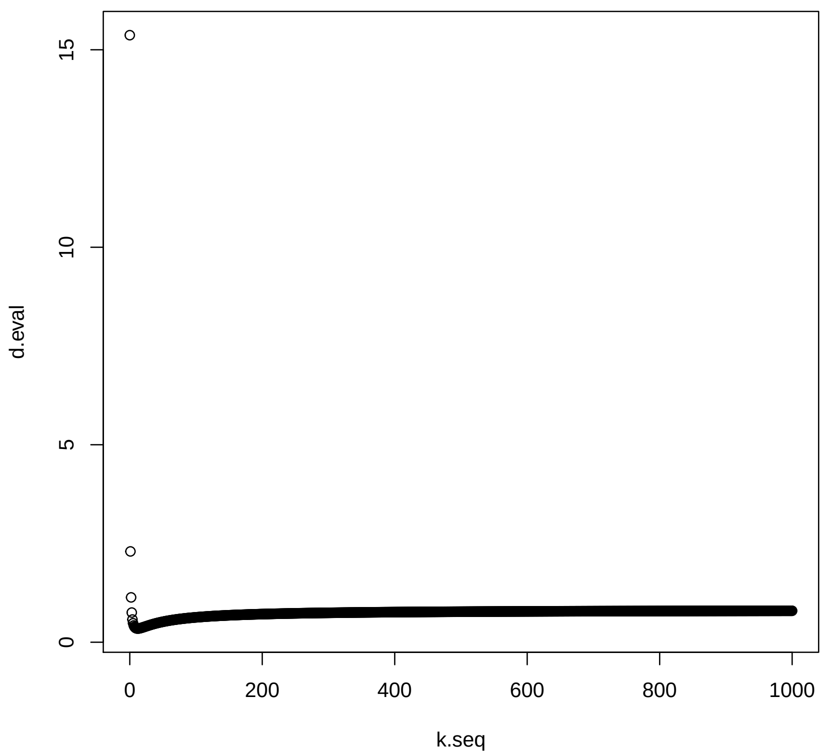
# 4 Problem 7

# 5 Problem 8

(a)

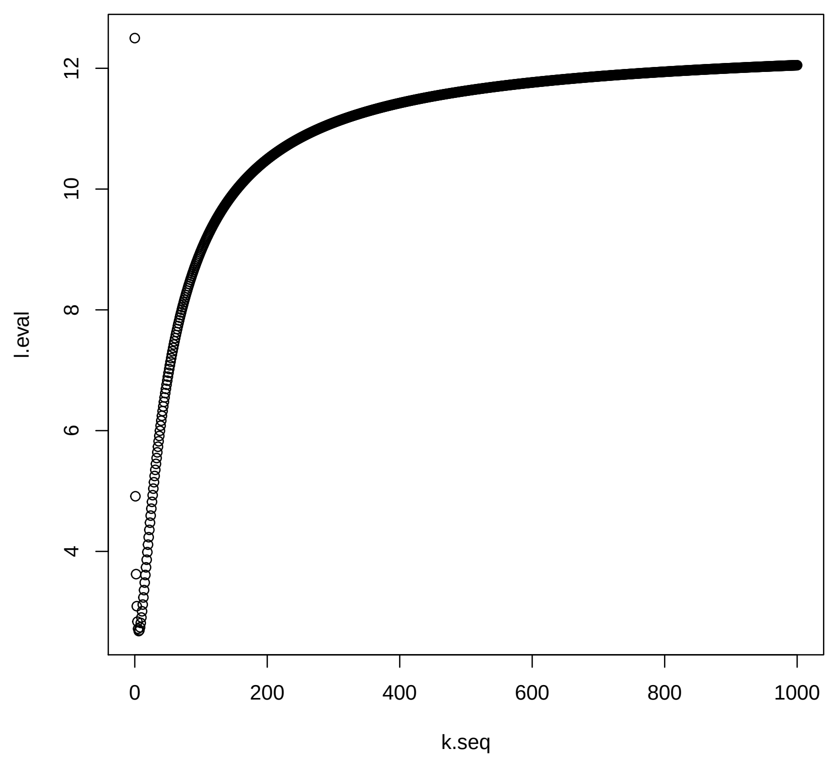
```
lambda = c(19, 3, 1, .7, .3)
e.beta = c(.8, .3, .2, .2, .1)
sig.sq = 2.5

k.seq = seq(0, 1000, 1)
```

```r
d.foo = function(k, sig.sq, lambda, e.beta)
{
  sig.sq * sum( lambda / (k+lambda)^2 ) +
    k^2 * sum( e.beta^2 / (k+lambda)^2 )
}

d.eval = sapply(k.seq, function(k)
  d.foo(k, sig.sq, lambda, e.beta))
plot(k.seq, d.eval)
```




```r
d.opt = optimize(d.foo, c(0, 100), sig.sq, lambda, e.beta); d.opt

## $minimum
## [1] 11.94841
##
## $objective
## [1] 0.346172
```

(b)
```
lambda = c(19, 3, 1, .7, .3)
e.beta = c(.8, .3, .2, .2, .1)
sig.sq = 2.5

k.seq = seq(0, 1000, 1)

l.foo = function(k, sig.sq, lambda, e.beta)
{
  sig.sq * sum( lambda^2 / (k+lambda)^2 ) +
    k^2 * sum( e.beta^2*lambda / (k+lambda)^2 )
}

l.eval = sapply(k.seq, function(k)
  l.foo(k, sig.sq, lambda, e.beta))
plot(k.seq, l.eval)
```



```r
l.opt = optimize(l.foo, c(0, 100), sig.sq, lambda, e.beta); l.opt

## $minimum
## [1] 6.179617
##
## $objective
## [1] 2.679957
```

(c)

```r
d_0 = d.foo(0, sig.sq, lambda, e.beta); d_0

## [1] 15.36967

d.opt

## $minimum
## [1] 11.94841
##
## $objective
## [1] 0.346172

l_0 = l.foo(0, sig.sq, lambda, e.beta); l_0

## [1] 12.5

l.opt

## $minimum
## [1] 6.179617
##
## $objective
## [1] 2.679957
```

For D(k), D(k.opt)<D(0), it's possible to improve over the ordinary least squares method using ridge regression. For L(k), L(k.opt)<L(0), which also means it's possible to improve over the ordinary least squares method using ridge regression.