

HW1

Kungang Zhang

January 23, 2017

Prob 1)

Answer:

Joint density:

$$f(\mathbf{y}; \mu, \sigma^2) \propto \frac{1}{\sigma^n} \exp\left\{-\frac{\sum_{i=1}^n (y_i - \mu)^2}{2\sigma^2}\right\}$$

We already know that the MLE of μ is that $\hat{\mu} = \bar{y}$. Take derivatives w.r.t σ and set it equal to 0, we get

$$-n/\sigma^{n+1} + \sum_{i=1}^n (y_i - \bar{y})^2 / \sigma^{n+3} = 0$$

So we get

$$\hat{\sigma} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2}$$

Prob 2)

```
rm(list = ls())
setwd("/Users/kungangzhang/Documents/OneDrive/Northwestern/Study/Courses/MSiA-420-0/HW1")
library(gdata)

## gdata: read.xls support for 'XLS' (Excel 97-2004) files ENABLED.
##

## gdata: read.xls support for 'XLSX' (Excel 2007+) files ENABLED.
##

## Attaching package: 'gdata'
```

```
## The following object is masked from 'package:stats':
##
##      nobs

## The following object is masked from 'package:utils':
##
##      object.size

df <- read.xls("./HW1_data.xls", sheet = 1, header = TRUE)
df <- df[, c("y", "x")]
set.seed(111)
```

(a) Linear model

Answer: coef equals to 29.62201 13.44881

```
mod1<-lm(I(y^(-1))~I(x^(-1)),data = df)
summary(mod1)

##
## Call:
## lm(formula = I(y^(-1)) ~ I(x^(-1)), data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.056684 -0.004123  0.000694  0.002766  0.063565
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.033759   0.006684   5.051 0.000118 ***
## I(x^(-1))    0.454014   0.020061  22.632 1.41e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02175 on 16 degrees of freedom
## Multiple R-squared:  0.9697, Adjusted R-squared:  0.9678
## F-statistic: 512.2 on 1 and 16 DF,  p-value: 1.411e-13

ga_mod1 <-
c(1/mod1$coefficients[1],mod1$coefficients[2]/mod1$coefficients[1])
ga_mod1

## (Intercept)    I(x^(-1))
##    29.62201    13.44881
```

(b) Nonlinear model

Answer: coef of nlm() equals to 28.13688 12.57428; coef of nls() equals to 28.13705 12.57445.

```
fn <- function(ga) {
  yhat <- ga[1] * df$x/(ga[2] + df$x)
  sum((df$y - yhat)^2)
}
out2 <- nlm(fn, p = ga_mod1, hessian = TRUE)
summary(out2)

##           Length Class  Mode
## minimum      1      -none- numeric
## estimate      2      -none- numeric
## gradient      2      -none- numeric
## hessian       4      -none- numeric
## code          1      -none- numeric
## iterations    1      -none- numeric

out2$estimate

## [1] 28.13688 12.57428

Fitfun <- function(x, ga) ga[1] * x/(ga[2] + x)
mod2 <- nls(y ~ Fitfun(x, ga), data = df, start = list(ga = ga_mod1))
summary(mod2)

##
## Formula: y ~ Fitfun(x, ga)
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## ga1  28.1370    0.7280    38.65  < 2e-16 ***
## ga2  12.5745    0.7631    16.48 1.85e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5185 on 16 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 4.347e-07

mod2$m$getPars()

##      ga1      ga2
## 28.13705 12.57445
```

Prob 3)

(a) Observed Fisher information matrix and the covariance matrix of the estimated parameter using the Hessian produced by `nlm()`. Calculate the standard errors of the estimated parameters.

Answer: SE for estimators are 0.7418084 0.7795476

```
#For Gaussian errors, the observed Fisher Information matrix equals to the  
Hessian(SSE)/2/MSE  
MSE <- out2$minimum/(length(df$y)-length(out2$estimate)) #estimate of the  
error variance  
InfoMat <- out2$hessian/2/MSE #observed information matrix  
CovTheta<-solve(InfoMat)  
SE<-sqrt(diag(CovTheta)) #standard errors of parameter estimates  
MSE  
  
## [1] 0.2688919  
  
CovTheta  
  
##           [,1]      [,2]  
## [1,] 0.5502797 0.5430248  
## [2,] 0.5430248 0.6076944  
  
SE  
  
## [1] 0.7418084 0.7795476
```

(b) Use `vcov()` applied to the output of `nls()` to calculate covariance and then standard errors.

Answer: SE for estimators are 0.7279790 0.7630534. It is similar to what we get from part (a), but smaller.

Question: How `vcov()` estimate the covariance of a model? Bootstrap?

```
cov_mod2<-vcov(mod2)  
SE_mod2 <- sqrt(diag(cov_mod2))  
SE_mod2  
  
##           ga1      ga2  
## 0.7279790 0.7630534
```

(c) Use part (a) and `confint.default()` applied to the output of `nls()` to get CIs of gamma, respectively, and compare

Answer: The 95% two-sided CI from part (a) is [26.76612 11.13378] (lower bound) and [29.50765 14.01479] (upper bound); the `confint.default(mod2)` gives [26.71024 11.07890] (lower bound) and [29.56386 14.07001] (upper bound)

Question: From the help file, "confint is a generic function. The default method assumes asymptotic normality, and needs suitable `coef` and `vcov` methods to be available." But why the results are different? The difference in minimizer?

```
#The part (a) is asymptotic results, assuming the normality
#Lower bound
out2$estimate+qnorm(0.025,0,1)*SE

## [1] 26.68296 11.04640

#upper bound
out2$estimate+qnorm(0.975,0,1)*SE

## [1] 29.59080 14.10217

confint.default(mod2)

##          2.5 %    97.5 %
## ga1 26.71024 29.56386
## ga2 11.07890 14.07001
```

Prob 4)

(a) Calculate and plot bootstrapped histograms of γ , and calculate bootstrapped SE

Answer: The histograms are the following plots. The SE of $\hat{\gamma}_0$ and $\hat{\gamma}_1$ are 0.7297492 0.7536710.

```
library(boot) #need to load the boot package
fit<-function(Z,i,ga0,x_pred) {
  #Z is the entire data set of replicates of bootstrapping data
  #i is for the i-th replicate
  #theta0 is the starting point
  #How to initialize during the bootstrapping?
```

```

Zboot<-Z[i,]
y<-Zboot[[1]];x<-Zboot[[2]];
fn <- function(ga) {yhat<-ga[1]*x/(ga[2]+x); sum((y-yhat)^2)}
out<-nlm(fn,p=ga0)
ga<-out$estimate #parameter estimates
y_pred<-ga[1]*x_pred/(ga[2]+x_pred)
c(ga,y_pred)}
Probbboot<-boot(df, fit, R=5000, ga0=ga_mod1,x_pred=27)#ga0 is just a constant
parameters used in the fit function
CovGa<-cov(Probbboot$t[,1:2])#Probbboot$t is just test-statistics of the
parameters we are interested in (default is the last parameter of the
function, here fit)
SE<-sqrt(diag(CovGa))
Probbboot

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = df, statistic = fit, R = 5000, ga0 = ga_mod1, x_pred = 27)
##
##
## Bootstrap Statistics :
##      original      bias      std. error
## t1*  28.13688 -0.09231511   0.7297492
## t2*  12.57428 -0.05989065   0.7536710
## t3*  19.19671 -0.03592575   0.2112369

CovGa

##           [,1]      [,2]
## [1,] 0.5325339 0.5123663
## [2,] 0.5123663 0.5680199

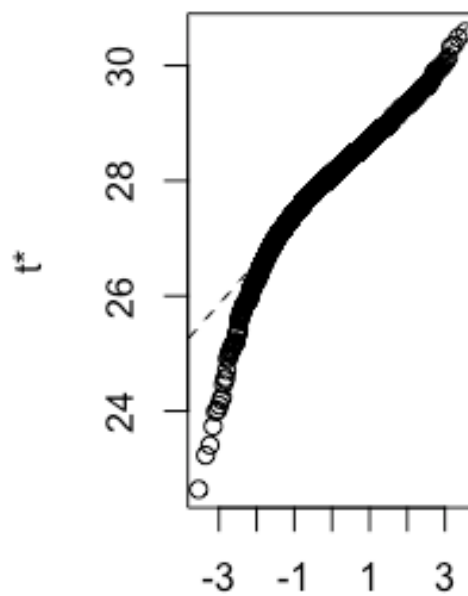
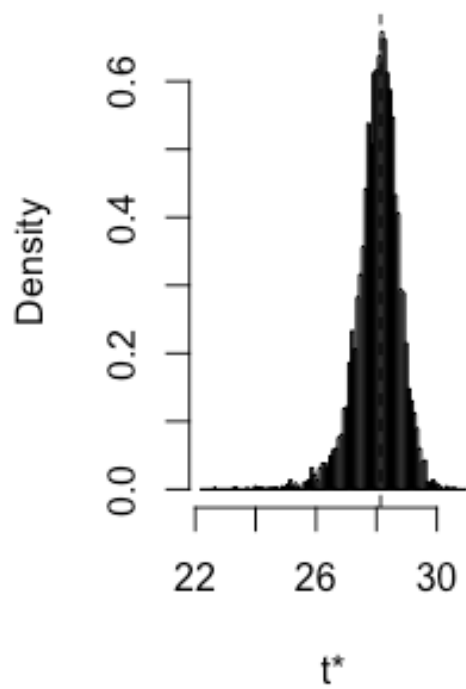
SE

## [1] 0.7297492 0.7536710

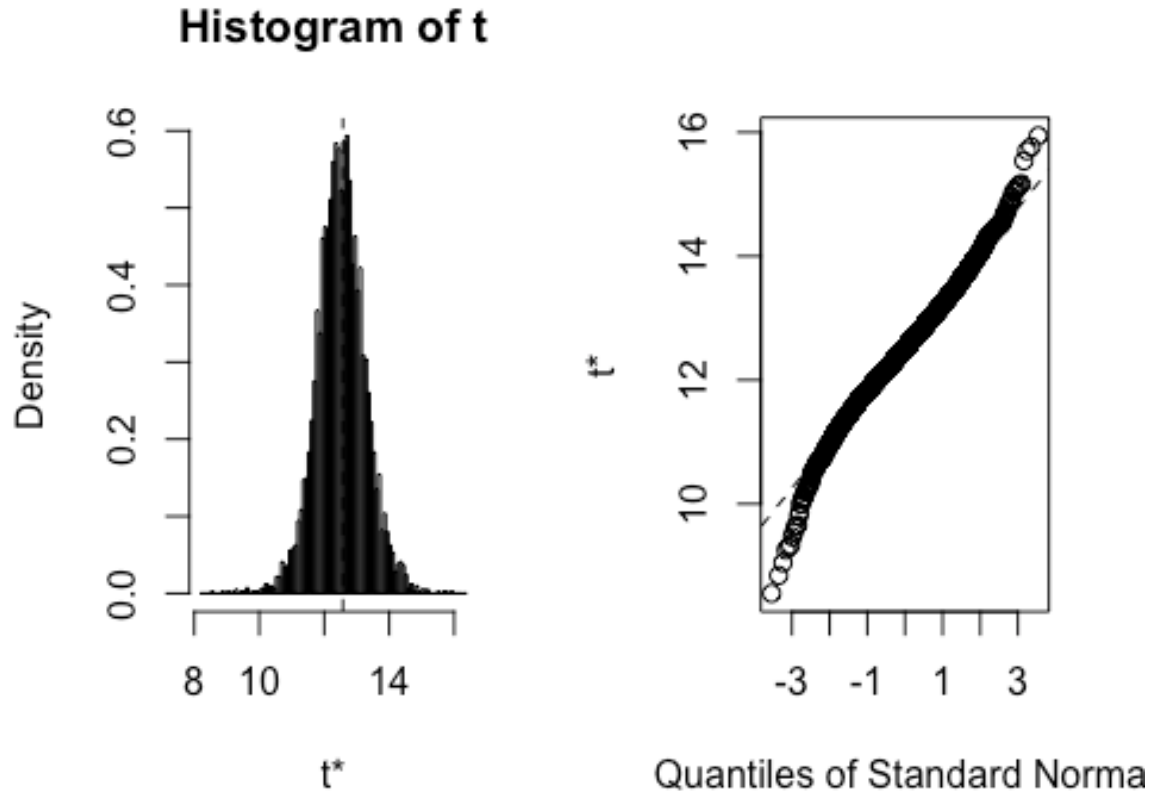
plot(Probbboot,index=1) #index=i calculates results for ith parameter

```

Histogram of t



```
plot(Proboot,index=2) #index=i calculates results for ith parameter
```



(b) Calculate "crude" two-sided 95% CIs using normal app

Answer: The crude: γ_0 (26.80, 29.66); γ_2 (11.16, 14.11).

```
boot.ci(Probboot,conf=c(.95),index=1,type=c("norm","basic"))#"norm" is the
shifted (shifted by biase) CI; "basic" is the reflected CI (should also be
shifted).
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
```

```
## Based on 5000 bootstrap replicates
```

```
##
```

```
## CALL :
```

```
## boot.ci(boot.out = Probboot, conf = c(0.95), type = c("norm",
##      "basic"), index = 1)
```

```
##
```

```
## Intervals :
```

```
## Level      Normal      Basic
```

```
## 95%   (26.80, 29.66 )  (26.97, 29.93 )
```

```
## Calculations and Intervals on Original Scale
```

```
boot.ci(Probboot,conf=c(.95),index=2,type=c("norm","basic"))#"norm" is the
shifted (shifted by biase) CI; "basic" is the reflected CI (should also be
shifted).
```



```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = Probbboot, conf = c(0.95), type = c("norm",
##      "basic"), index = 2)
##
## Intervals :
## Level      Normal      Basic
## 95%    (11.16, 14.11 )  (11.12, 14.14 )
## Calculations and Intervals on Original Scale
```

(c) Calculate reflected two-sided 95% CIs using normal app

Answer: The reflected: γ_0 (26.97, 29.93); γ_0 (11.12, 14.14).

(d) Do the CIs in part (c) agree with those in part (b)?

Answer: Yes, they are pretty close, because the histograms are very close to normal.

Prob 5)

The CI's of predictable part and prediction interval. And compare them.

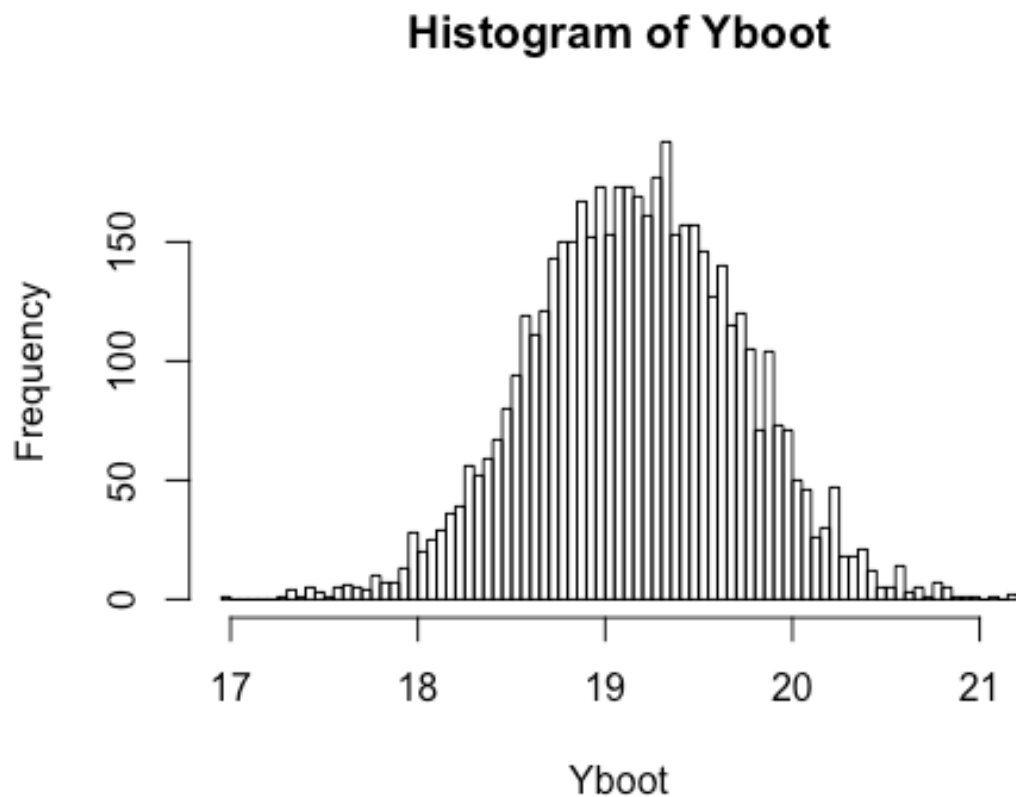
Answer: The CI for predictable part: (18.91, 19.74); The PI: (18.14896 20.33852). The PI is wider. PI should be expected to contain the future response with roughly 95% chance, because the future response contains error and PI has counted for that.

#CI for predictable part:
boot.ci(Probbboot, conf=c(.95), index=3, type=c("norm", "basic")) *#"norm" is the shifted (shifted by biase) CI; "basic" is the reflected CI (should also be shifted).*

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = Probbboot, conf = c(0.95), type = c("norm",
##      "basic"), index = 3)
##
## Intervals :
## Level      Normal      Basic
```

```
## 95% (18.82, 19.65 ) (18.91, 19.74 )
## Calculations and Intervals on Original Scale

#Prediction interval:
e<-rnorm(nrow(Probboot$t), mean=0, sd=sqrt(MSE))
Yboot<-Probboot$t[,3]-e#predicted response
Yquant<-quantile(Yboot,prob=c(.025,.975))
L<-2*Probboot$t0[3]-Yquant[2]
U<-2*Probboot$t0[3]-Yquant[1]
hist(Yboot,100)
```



```
c(L,U) #more complex PI

## 97.5% 2.5%
## 18.14896 20.33852
```

Prob 6)

Use AIC criterion to compare models from Prob 2)(b) and from Prob 6)

Answer: The AIC for the two models are 1.628871 and 2.836148, so that AIC criterion suggests to take the first model.

```
n=nrow(df)
FitFun1 <- function(x,ga) ga[1]*x/(ga[2]+x)
out1<-nls(y~FitFun1(x,ga),data=df,start=list(ga=ga_mod1))
summary(out1)

##
## Formula: y ~ FitFun1(x, ga)
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## ga1  28.1370      0.7280   38.65  < 2e-16 ***
## ga2  12.5745      0.7631   16.48 1.85e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5185 on 16 degrees of freedom
##
## Number of iterations to convergence: 4
## Achieved convergence tolerance: 4.347e-07

AIC1<- -2*as.numeric(logLik(out1))/n+2*2/n
AIC(out1)#The in-built AIC function has formula -LogLik(out)+2*(p+1)

## [1] 31.31967

FitFun2 <- function(x,ga) ga[1]+ga[2]*sqrt(x)
out2<-nls(y~FitFun2(x,ga),data=df,start=list(ga=ga_mod1))
summary(out2)

##
## Formula: y ~ FitFun2(x, ga)
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## ga1  -0.4566      0.5154  -0.886    0.389
## ga2   3.7720      0.1401  26.918 9.41e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9483 on 16 degrees of freedom
##
## Number of iterations to convergence: 1
## Achieved convergence tolerance: 3.418e-07
```

```
AIC2<- -2*as.numeric(logLik(out2))/n+2*2/n
AIC(out2)#The in-built AIC function has formula -LogLik(out)+2*(p+1)
## [1] 53.05066
```

Prob 7)

Use n-fold CV to compare models from Prob 2)(b) and from Prob 6)

Answer: Basically I used different forms and R-functions to do the CV. The first two MSEs are based on nlm() for the two models; the third MSE is based on lm() for the second model; the last two MSEs are based on nls() for the two model. Those MSEs corresponding to the same model are slightly different probably because the minimizers are different. The MSE vector is [0.29430331.11065381.11065520.29430151.1106553] (the first and the fourth corresponds to the model from Prob 2)(b)). So CV suggests we should use the model from Prob 2)(b).

```
#R commands for creating indices of partition for K-fold CV
CVInd <- function(n,K) {
  #n is sample size; K is number of parts; returns K-length list of indices
for each part
  m<-floor(n/K) #approximate size of each part
  r<-n-m*K
  I<-sample(n,n) #random reordering of the indices
  Ind<-list() #will be list of indices for all K parts
  length(Ind)<-K
  for (k in 1:K) {
    if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
    else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
    Ind[[k]] <- I[kpart] #indices for kth part of data
  }
  Ind
}

#Shell for running multiple random replicates of CV
Nrep<-1 #number of replicates of CV
n=nrow(df)
K<-n #K-fold CV on each replicate
n.models = 5 #number of different models to fit and compare
y<-df$y
x<-df$x
yhat=matrix(0,n,n.models)
```

```

MSE_cv<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {#ALL models use exactly the same CV partition
    y_temp = y[-Ind[[k]]]
    x_temp = x[-Ind[[k]]]

    fn1 <- function(ga) {yhat_temp<-ga[1]*x_temp/(ga[2]+x_temp); sum((y_temp-
yhat_temp)^2)}#the first model to compare
    out<-nlm(fn1,p=ga_mod1,hessian=TRUE)
    ga_temp<-out$estimate #parameter estimates
    yhat[Ind[[k]],1]<-ga_temp[1]*x[Ind[[k]]]/(ga_temp[2]+x[Ind[[k]]])#The
prediction part doesn't include random error
    fn2 <- function(ga) {yhat_temp<-ga[1]+ga[2]*sqrt(x_temp); sum((y_temp-
yhat_temp)^2)}#the second model to compare
    out<-nlm(fn2,p=ga_mod1,hessian=TRUE)
    ga_temp<-out$estimate #parameter estimates
    yhat[Ind[[k]],2]<-ga_temp[1]+ga_temp[2]*sqrt(x[Ind[[k]]])#The prediction
part doesn't include random error
    out<-lm(y~I(x^(.5)),df[-Ind[[k]],]) #the second model to compare, just
using different function to calculate, because it can be looked as a linear
model. Should be the same as previous.
    yhat[Ind[[k]],3]<-as.numeric(predict(out,df[Ind[[k]],]))
    #Using nls() function to redo the CV for model 1 and model 2
    out<-nls(y~FitFun1(x,ga),data=df[-Ind[[k]],],start=list(ga=ga_mod1))
    yhat[Ind[[k]],4]<-as.numeric(predict(out,df[Ind[[k]],]))
    out<-nls(y~FitFun2(x,ga),data=df[-Ind[[k]],],start=list(ga=ga_mod1))
    yhat[Ind[[k]],5]<-as.numeric(predict(out,df[Ind[[k]],]))
  } #end of k Loop
  MSE_cv[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j Loop
MSE_cv

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.2943033 1.110654 1.110655 0.2943015 1.110655

MSEAve_cv<- apply(MSE_cv,2,mean); MSEAve_cv #averaged mean square CV error
## [1] 0.2943033 1.1106538 1.1106552 0.2943015 1.1106553

MSEsd_cv <- apply(MSE_cv,2,sd); MSEsd_cv #SD of mean square CV error
## [1] NA NA NA NA NA

r2<-1-MSEAve_cv/var(y); r2 #CV r^2
## [1] 0.9924874 0.9716485 0.9716484 0.9924874 0.9716484

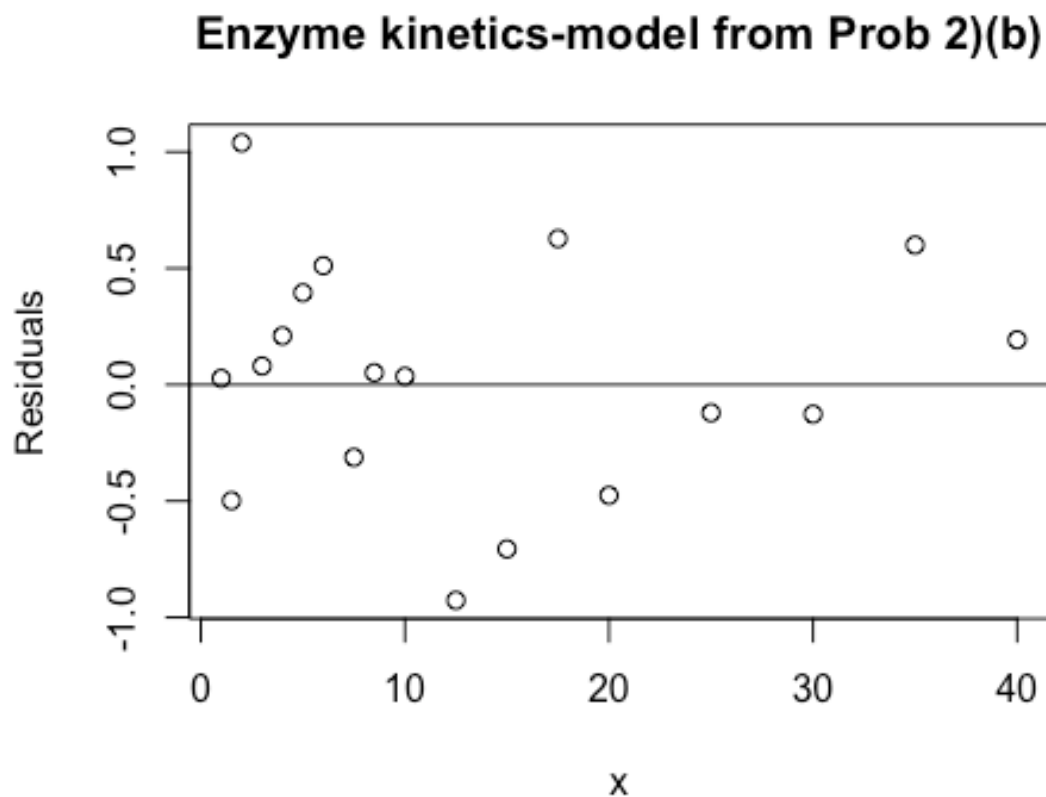
```

Prob 8)

Use residual plots to determine which models we used to compare in Prob 6) and 7) is better.

Answer: Obvious, the residual of model from Prob 6) is not independent of x , while the residual of model from Prob 2)(b) is more independent of x . So it suggests that the model from Prob 2) is better and it is consistent with the conclusions from Prob 6) and Prob 7).

```
#Residual plot for the model from Prob 2)(b) (nls() model with the entire dataset, out1)
plot(df$x, resid(out1), ylab="Residuals", xlab="x", main="Enzyme kinetics-model from Prob 2)(b)")
abline(0, 0)
```



```
#Residual plot for the model from Prob 6 (nls() model with the entire dataset, out2)
plot(df$x, resid(out2), ylab="Residuals", xlab="x", main="Enzyme kinetics-model from Prob 6)")
```

```
from Prob 6)")  
abline(0, 0)
```

