Notes 6: Regression Diagnostics (Part A)

Nathaniel E. Helwig

Department of Statistics
University of Illinois at Urbana-Champaign

Stat 420: Methods of Applied Statistics
Section N1U/N1G – Spring 2014

Outline of Notes

1) MLR Review:
- Model form
- Model assumptions
- OLS estimation

2) Normality Assumption:
- Visualizing non-normality
- Testing for non-normality
- Solutions for non-normality

3) Constant σ^2 Assumption:
- Visualizing non-constant σ^2
- Testing for non-constant σ^2
- Solutions for non-constant σ^2

4) Equal Influence Assumption:
- Visualizing unequal influence
- Testing for unequal influence
- Solutions for unequal influence

MLR Model: Form (scalar)

The multiple linear regression model has the form

$$y_i = b_0 + \sum_{j=1}^{p} b_j x_{ij} + e_i$$

for $i \in \{1, \ldots, n\}$ where

- $y_i \in \mathbb{R}$ is the real-valued response for the i-th observation
- $b_0 \in \mathbb{R}$ is the regression intercept
- $b_j \in \mathbb{R}$ is the j-th predictor's regression slope
- $x_{ij} \in \mathbb{R}$ is the j-th predictor for the i-th observation
- $e_i \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma^2)$ is Gaussian measurement error

MLR Model: Form (matrix)

The multiple linear regression model has the form

$$\mathbf{y} = \mathbf{Xb} + \mathbf{e}$$

or

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ 1 & x_{31} & x_{32} & \cdots & x_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_n \end{pmatrix}$$

MLR Assumptions: (scalar form)

The fundamental assumptions of the MLR model are:

1. Relationship between x_j and y is linear (given other predictors)
2. x_{ij} and y_i are observed random variables (constants)
3. $e_i \overset{\text{iid}}{\sim} \text{N}(0, \sigma^2)$ is an unobserved random variable
4. $b_0, b_1, \ldots, b_p$ are unknown constants
5. $(y_i | x_{i1}, \ldots, x_{ip}) \overset{\text{ind}}{\sim} \text{N}(b_0 + \sum_{j=1}^{p} b_j x_{ij}, \sigma^2)$
   note: homogeneity of variance

Note: b_j is expected increase in Y for 1-unit increase in X_j with all other predictor variables held constant

MLR Assumptions (matrix form)

In matrix terms, the error vector is multivariate normal:

$$\mathbf{e} \sim \mathrm{N}(\mathbf{0}_n, \sigma^2 \mathbf{I}_n)$$

In matrix terms, the response vector is multivariate normal given $\mathbf{X}$:

$$(\mathbf{y}|\mathbf{X}) \sim \mathrm{N}(\mathbf{X}\mathbf{b}, \sigma^2 \mathbf{I}_n)$$

Ordinary Least Squares

The ordinary least squares (OLS) problem is

$$\min_{\mathbf{b} \in \mathbb{R}^{p+1}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2$$

where $\|\cdot\|$ denotes the Frobenius norm.

The OLS solution has the form

$$\hat{\mathbf{b}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

which is the same formula from SLR!

Fitted Values and Residuals

SCALAR FORM:

Fitted values are given by

$$\hat{y}_i = \hat{b}_0 + \sum_{j=1}^{p} \hat{b}_j x_{ij}$$

and *residuals* are given by

$$\hat{e}_i = y_i - \hat{y}_i$$

MATRIX FORM:

Fitted values are given by

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{b}} = \mathbf{H}\mathbf{y}$$

and *residuals* are given by

$$\hat{\mathbf{e}} = \mathbf{y} - \hat{\mathbf{y}} = (\mathbf{I}_n - \mathbf{H})\mathbf{y}$$

Estimated Error Variance (Mean Squared Error)

The estimated error variance is

$$\begin{aligned}\hat{\sigma}^2 &= SSE/(n-p-1) \\ &= \sum_{i=1}^{n}(y_i - \hat{y}_i)^2/(n-p-1) \\ &= \|(\mathbf{I}_n - \mathbf{H})\mathbf{y}\|^2/(n-p-1)\end{aligned}$$

which is an unbiased estimate of error variance σ^2.

The estimate $\hat{\sigma}^2$ is the *mean squared error (MSE)* of the model.

Distribution of Estimator, Fitted Values, and Residuals

Remember the MLR assumptions imply that

$$\hat{\mathbf{b}} \sim \mathrm{N}(\mathbf{b}, \sigma^2(\mathbf{X}'\mathbf{X})^{-1})$$

$$\hat{\mathbf{y}} \sim \mathrm{N}(\mathbf{X}\mathbf{b}, \sigma^2\mathbf{H})$$

$$\hat{\mathbf{e}} \sim \mathrm{N}(\mathbf{0}, \sigma^2(\mathbf{I}_n - \mathbf{H}))$$

Typically σ^2 is unknown, so we use the MSE $\hat{\sigma}^2$ in practice.

Visualizing Non-Normality: Overview

Two visualizations (plots) useful for examining normality:

- QQ-plot: plots empirical (estimated) quantiles against theoretical normal quantiles
- Histogram: plots empirical (estimated) distribution of data

It is often helpful to add references lines to the plots:

- QQ-plot: add 45° line and/or qq-line (i.e., quantile-quantile line)
- Histogram: add empirical density for MLE normal

QQ-Plot: Definition

Empirical quantiles are estimated by sorting the x_i values:

$$q_i = x_{(i)}$$

where q_i is the i-th sample quantile and $x_{(i)}$ is the i-th order statistic (so that $x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n)}$).

Theoretical quantiles are estimated using theoretical percentiles:

$$t_i = \frac{\tilde{q}_i - c}{n + 1 - 2c}$$

where $\tilde{q}_i \in \{1, \ldots, n\}$ is the i-th input quantile and $0 < c < 1$ is a correction factor; R sets $c = 1/2$ if $n > 10$ and $c = 3/8$ if $n \leq 10$.

QQ-Plot: R Function

We can design our own QQ-plot function:

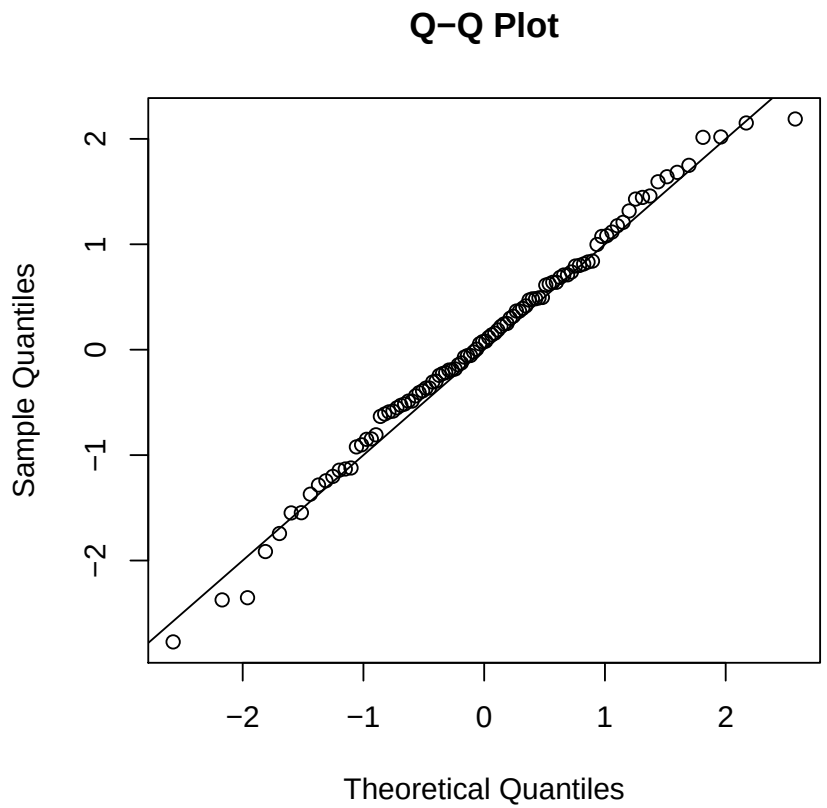
```
myqqplot <- function(x,mydist=qnorm,
                     xlim=NULL,ylim=NULL,...){
  yval=sort(x)
  nx=length(x)
  ac=ifelse(nx<=10,3/8,1/2)
  cs=((1:nx)-ac)/(nx+1-2*ac)
  xval=mydist(cs,...)
  plot(xval,yval,xlab="Theoretical Quantiles",
       ylab="Sample Quantiles",main="Q-Q Plot",
       xlim=xlim,ylim=ylim)
  abline(0,1)
}
```

Note: this is similar to R's built-in `qqplot` function, but adds a 45° line.

QQ-Plot: Example #1 (Normal Data)

```
> set.seed(773)
> x=rnorm(100)
> myqqplot(x)
```

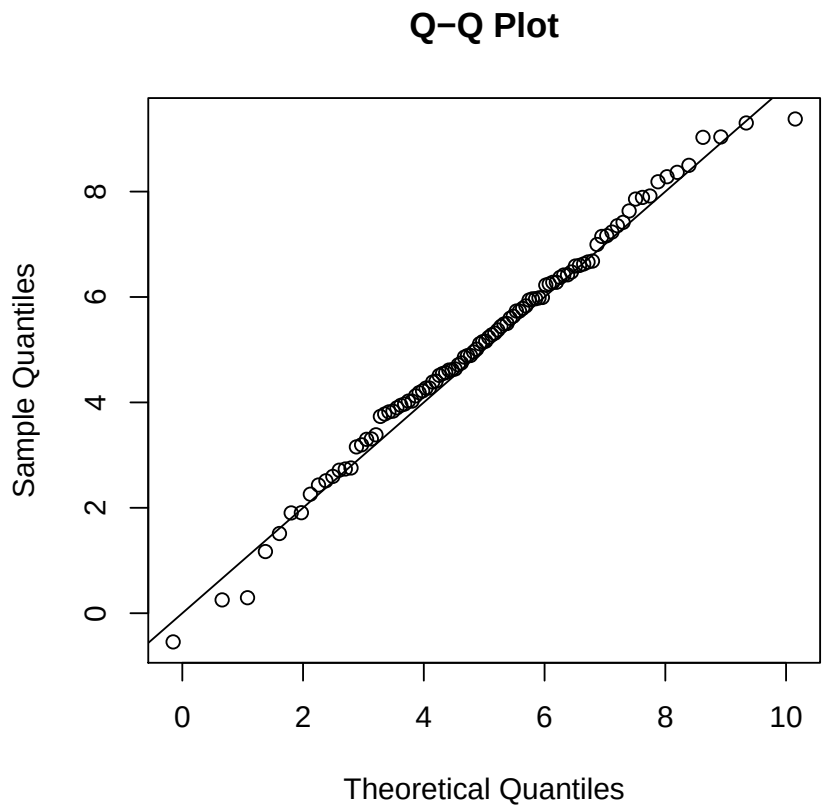
Simulate standard normal data and use QQ plot with theoretical quantiles from $\mathrm{N}(0, 1)$ distribution.



QQ-Plot: Example #2 (Normal Data)

```
> set.seed(773)
> x=rnorm(100,mean=5,sd=2)
> myqqplot(x,mean=5,sd=2)
```

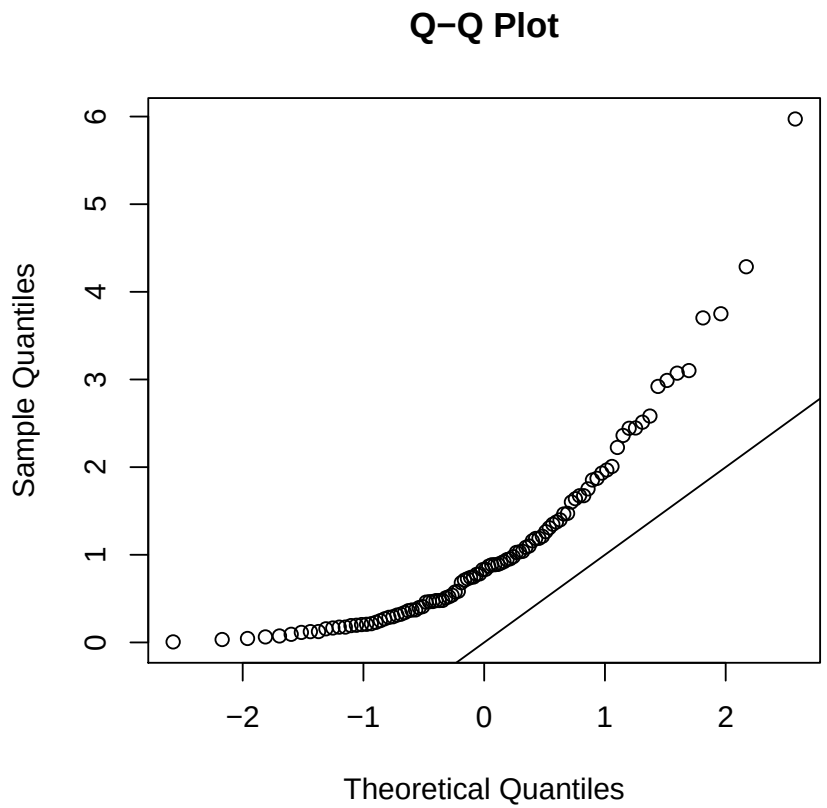
Simulate $\mathrm{N}(5, 4)$ data and use QQ plot with theoretical quantiles from $\mathrm{N}(5, 4)$ distribution.



QQ-Plot: Example #3 (Non-Normal Data)

```
> set.seed(773)
> x=rexp(100)
> myqqplot(x)
```

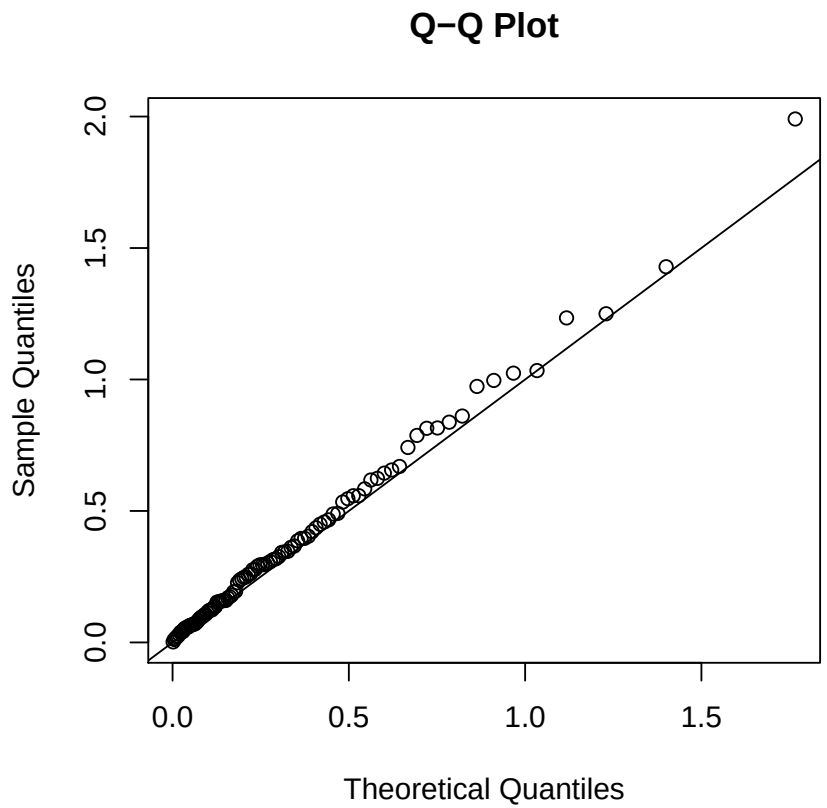
Simulate $\text{Exp}(1)$ data and use QQ plot with theoretical quantiles from $\text{N}(0, 1)$ distribution.



QQ-Plot: Example #4 (Non-Normal Data)

```
> set.seed(773)
> x=rexp(100,rate=3)
> myqqplot(x,qexp,rate=3)
```

Simulate $\text{Exp}(3)$ data and use QQ plot with theoretical quantiles from $\text{Exp}(3)$ distribution.



Histogram: Definition

Given $\mathbf{x}_i$ for $i \in \{1, \ldots, n\}$, a histogram bins (groups) the sample data and plots the frequency (count) of the number of sample observations falling within each bin. More specifically,

$$h_j = \sum_{i=1}^{n} I_{\{a_j \leq x_i < b_j\}}$$

where h_j is the height of the j-th bin, a_j and b_j are the lower and upper limits of the j-th bin, and $I_{\{\cdot\}}$ denotes an indicator function.

When examining normality, it is helpful to overlay the best-fitting normal density on the histogram.

Histogram: R Function

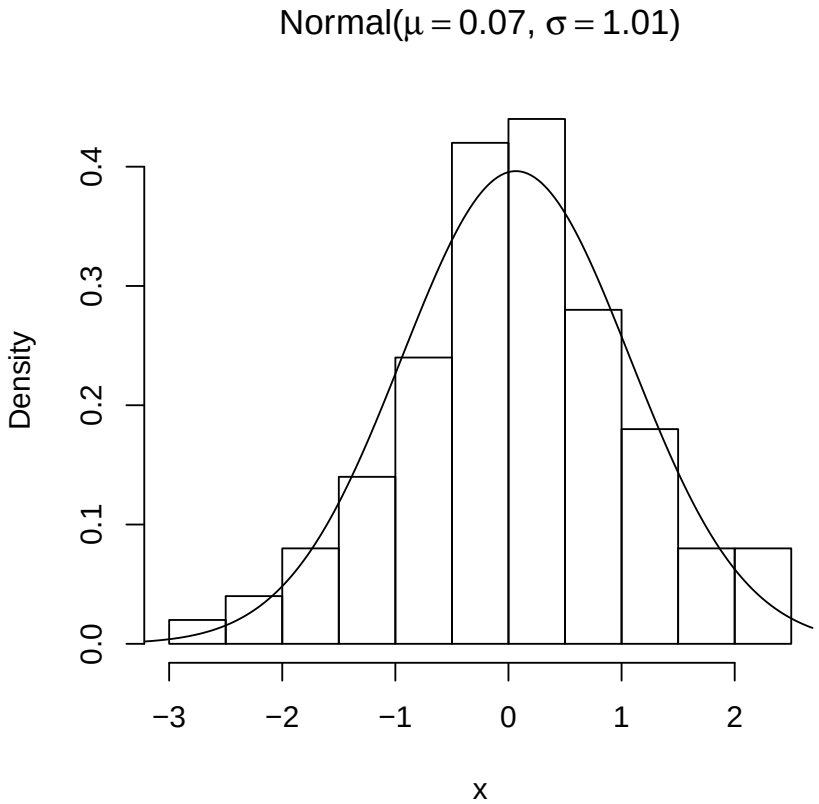
We can design our own histogram normal density function:

```
hnorm <- function(x,mu=NULL,sigma=NULL,main=NULL,
                  ndig=rep(2,2),cf=rep(1/2,2),ndp=200,...){
  if(is.null(mu)){mu=mean(x)}
  if(is.null(sigma)){nx=length(x); sigma=sqrt(var(x)*(nx-1)/nx)}
  if(is.null(main)){
    t1=bquote("Normal("*mu==.(round(mu,digits=ndig[1])))
    t2=bquote(sigma==.(round(sigma,digits=ndig[2]))*")")
    main=bquote(paste(.(t1),", ",.(t2),sep=""))
  }
  xseq=seq(min(x)-cf[1],max(x)+cf[2],length.out=ndp)
  xden=dnorm(xseq,mean=mu,sd=sigma)
  xhis=hist(x,plot=FALSE)
  ylim=range(xden); ylim[2]=max(max(xhis$density),ylim[2])
  hist(x,freq=FALSE,main=main,ylim=ylim)
  lines(xseq,xden,...)
}
```

Histogram: Example #1 (Normal Data)

```
> set.seed(773)
> x=rnorm(100)
> hnorm(x)
```

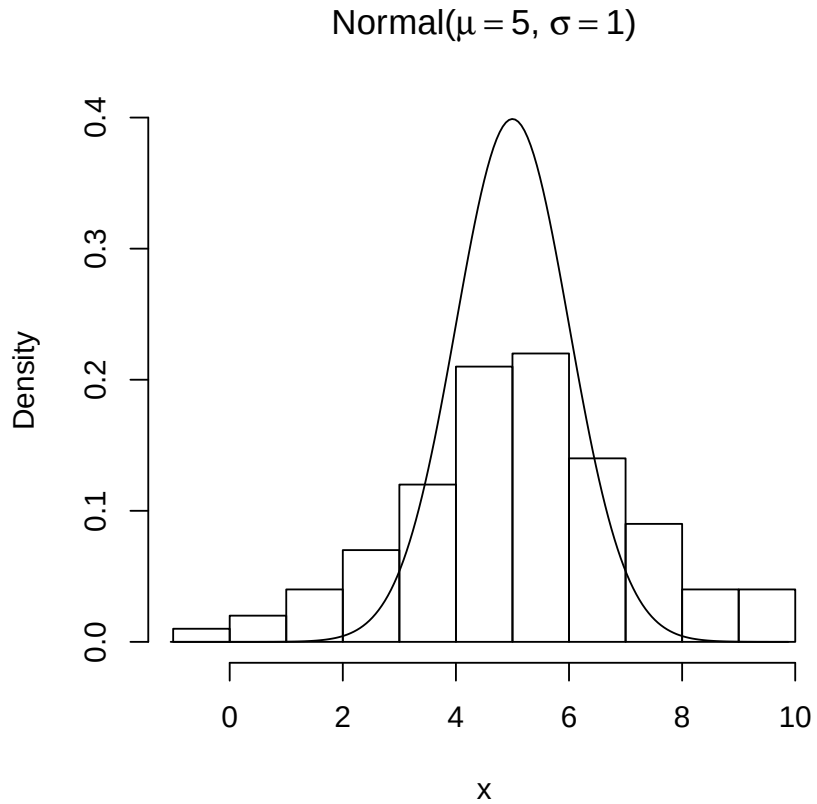
Simulate $\mathrm{N}(0, 1)$ data and use histogram with MLE normal density overlaid.



Histogram: Example #2 (Normal Data)

```
> set.seed(773)
> x=rnorm(100,mean=5,sd=2)
> hnorm(x,mu=5,sigma=1)
```

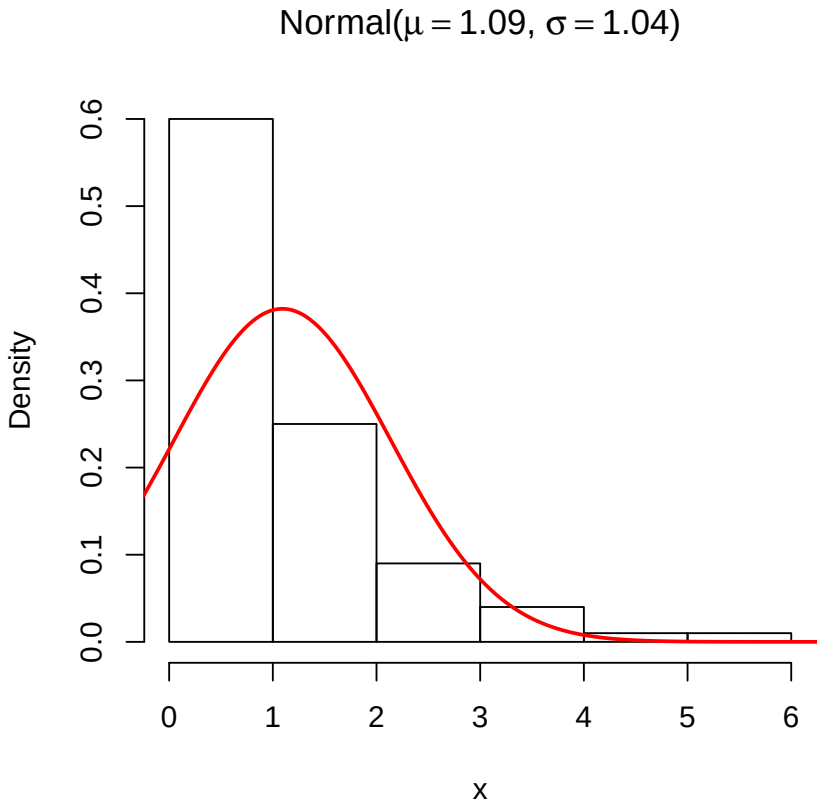
Simulate $\mathrm{N}(5, 4)$ data and use histogram with $\mathrm{N}(5, 1)$ density overlaid.



Histogram: Example #3 (Non-Normal Data)

```
> set.seed(773)
> x=rexp(100)
> hnorm(x,col="red",lwd=2)
```

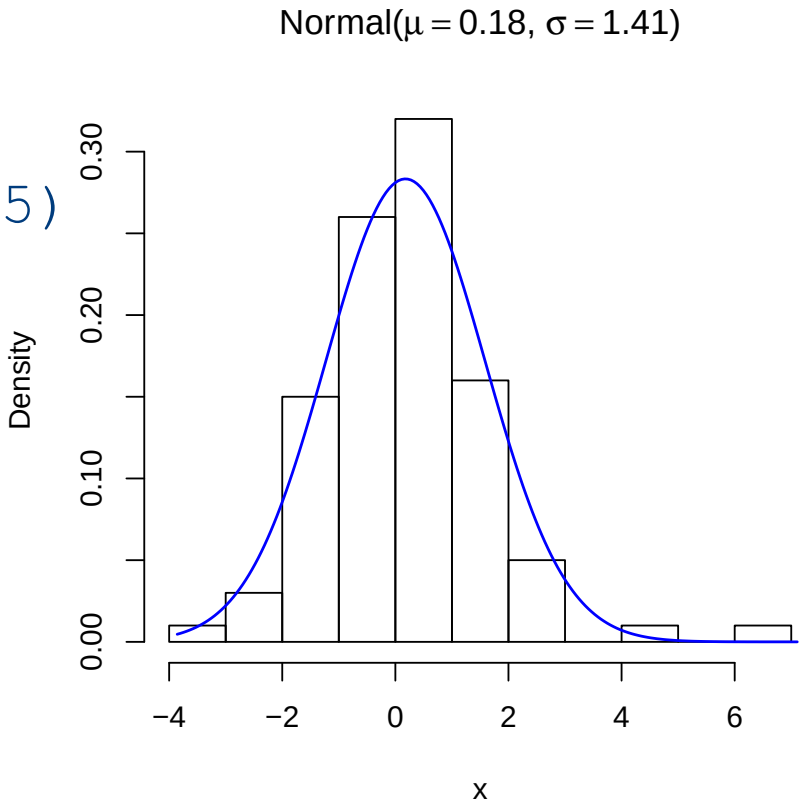
Simulate $\text{Exp}(1)$ data and use histogram with MLE normal density overlaid.



Histogram: Example #1 (Non-Normal Data)

```
> set.seed(773)
> x=rt(100,df=3)
> hnorm(x,col="blue",lwd=1.5)
```

Simulate t_3 data and use histogram with MLE normal density overlaid.



Testing for Non-Normality: Overview

QQ-plots and histograms provide nice visualizations, but are not formal tests of whether X follows a normal distribution.

To formally test the normality assumption, we could use:

- Looney-Gulledge (LG) correlation test
- Shapiro-Wilk (SW) normality test

LG Test: Definition

Looney-Gulledge (LG) correlation test computes the correlation between observed and expected quantiles.

LG test approximates the theoretical normal quantiles using:

$$z_i = \Phi^{-1}\left(\frac{r_i - 0.375}{n + 0.25}\right)$$

where $r_i \in \{1, \ldots, n\}$ is the i-th rank and Φ is standard normal CDF.

Looney, S. W., & Gulledge, T. R. (1985). Use of the correlation coefficient with normal probability plots. *The American Statistician, 39*, 75–79.

LG Test: Definition (continued)

Test is $H_0 : \rho_{LG} = 1$ versus $H_0 : \rho_{LG} < 1$, where ρ_{LG} is the population correlation coefficient between the observed data quantiles and the corresponding theoretical normal quantiles.

Use Monte Carlo methods:

- Calculate LG correlation for observed sample
- Calculate LG correlation for MANY random samples, where data is sampled with H_0 true
- Compare observed LG correlation to distribution of correlations

LG Test: R Functions

We can design our own LG correlation and normality test functions:

```
LGcor<-function(x){
  z=qnorm((rank(x)-0.375)/(length(x)+0.25))
  cor(x,z)
}

LGnormtest<-function(x,nsamp=10000,alpha=0.05,plot=FALSE){
  nx=length(x)
  mcsamp=replicate(nsamp,LGcor(rnorm(nx)))
  mycor=LGcor(x)
  pval=sum(mcsamp<mycor)/nsamp
  cval=quantile(mcsamp,alpha)
  if(plot){
    # some omitted code to plot data (see next slide)
  }
  list(rho=mycor,cval=cval,pval=pval)
}
```

LG Test: R Functions (continued)

Full normality test function with plotting R code:

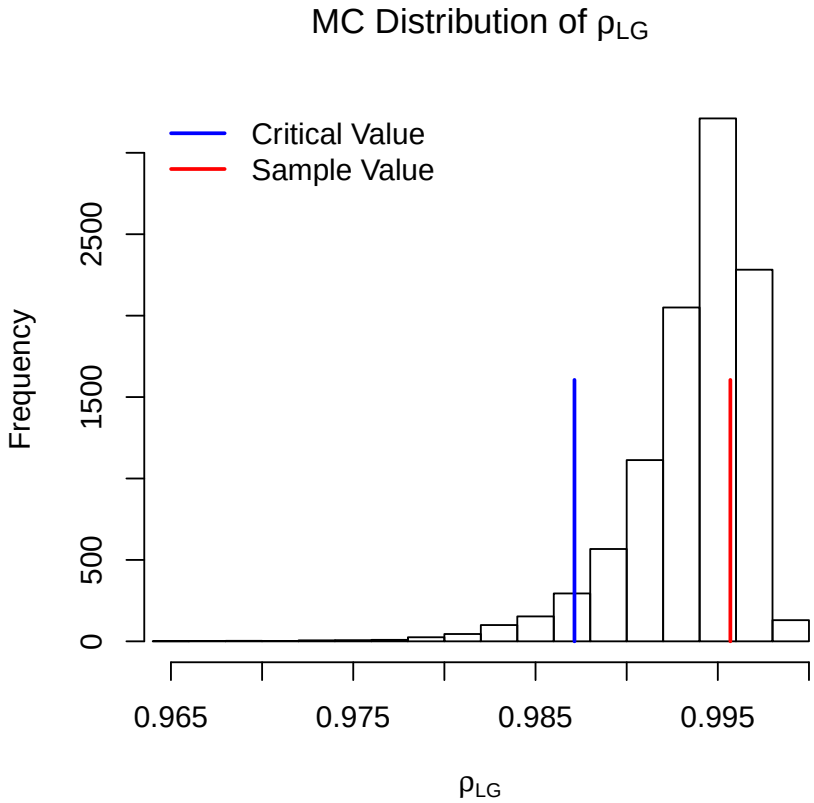
```
LGnormtest<-function(x,nsamp=10000,alpha=0.05,plot=FALSE){
  nx=length(x)
  mcsamp=replicate(nsamp,LGcor(rnorm(nx)))
  mycor=LGcor(x)
  pval=sum(mcsamp<mycor)/nsamp
  cval=quantile(mcsamp,alpha)
  if(plot){
    y=hist(mcsamp,plot=FALSE)
    xlim=range(mcsamp)
    if(mycor<xlim[1]){xlim[1]=mycor} else if(mycor>xlim[2]){xlim[2]=mycor}
    hist(mcsamp,main=expression("MC Distribution of "*rho[LG]),
         xlab=expression(rho[LG]),xlim=xlim)
    lh=c(0,max(y$counts)*.5)
    lines(rep(cval,2),lh,col="blue",lwd=2)
    lines(rep(mycor,2),lh,col="red",lwd=2)
    legend("topleft",c("Critical Value", "Sample Value"),lty=c(1,1),
           lwd=rep(2,2),col=c("blue","red"),bty="n")
  }
  list(rho=mycor,cval=cval,pval=pval)
}
```

LG Test: Example #1 (Normal Data)

```
> set.seed(773)
> x=rnorm(100)
> LGnormtest(x,plot=TRUE)
$rho
[1] 0.9956885

$cval
       5%
0.9871361

$pval
[1] 0.7042
```

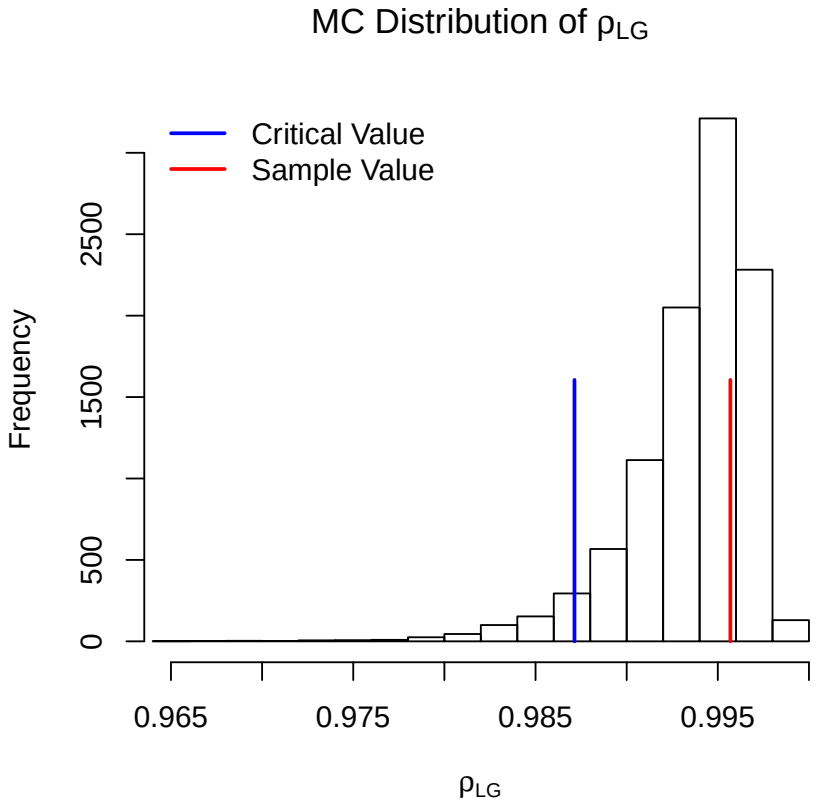


LG Test: Example #2 (Normal Data)

```
> set.seed(773)
> x=rnorm(100,mean=5,sd=2)
> LGnormtest(x,plot=TRUE)
$rho
[1] 0.9956885

$cval
       5%
0.9871361

$pval
[1] 0.7042
```

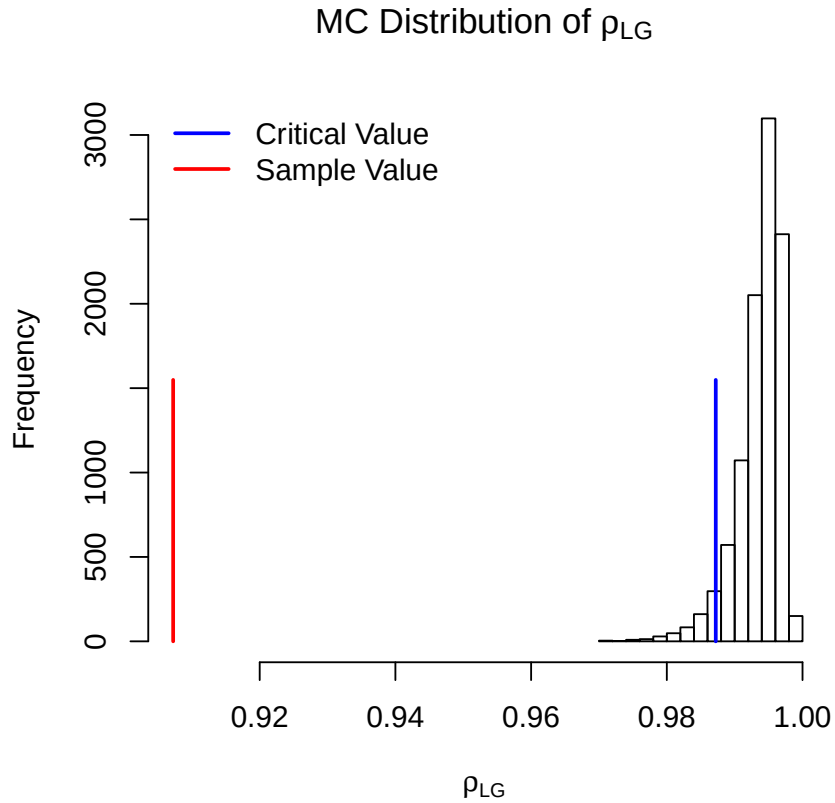


LG Test: Example #3 (Non-Normal Data)

```
> set.seed(773)
> x=rexp(100)
> LGnormtest(x,plot=TRUE)
$rho
[1] 0.9072432

$cval
       5%
0.9872141

$pval
[1] 0
```

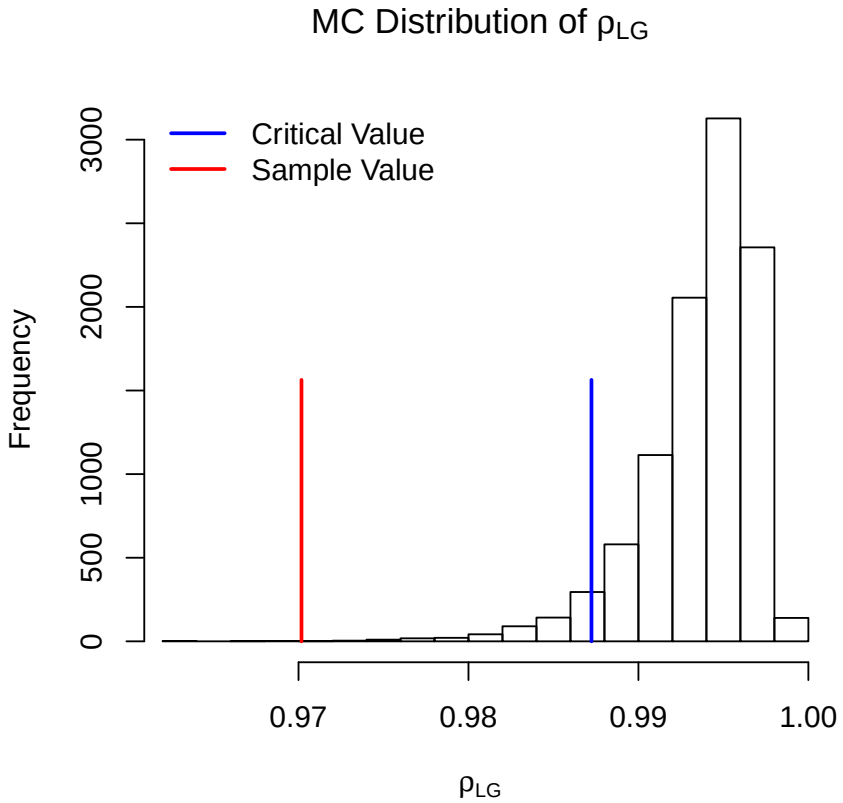


LG Test: Example #4 (Non-Normal Data)

```
> set.seed(773)
> x=rt(100,3)
> LGnormtest(x,plot=TRUE)
$rho
[1] 0.9701741

$cval
       5%
0.9872367

$pval
[1] 5e-04
```

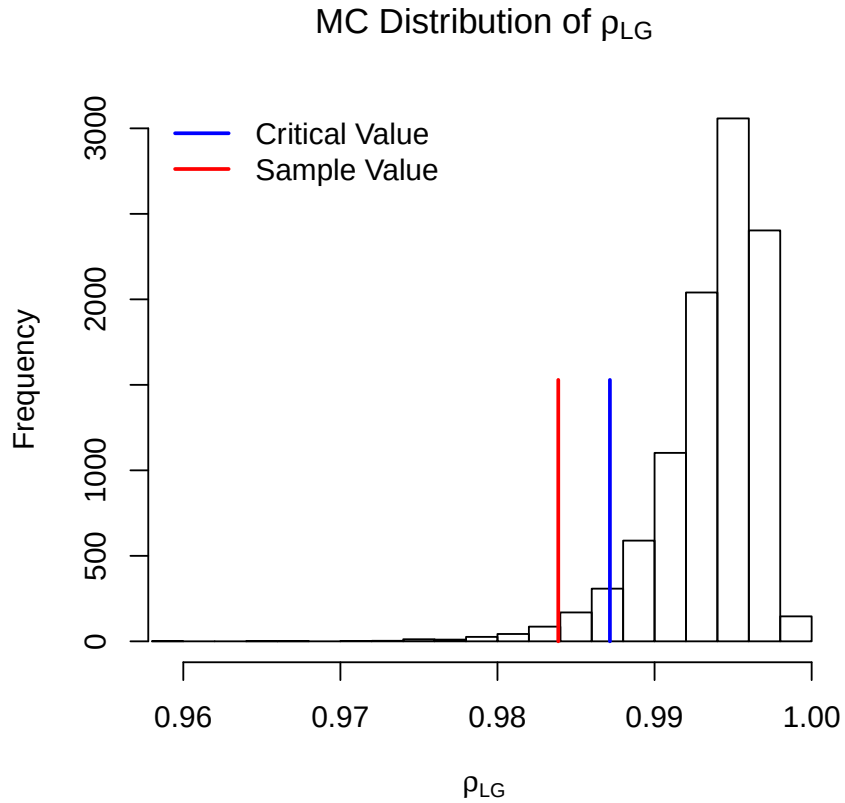


LG Test: Example #5 (Almost-Normal Data)

```
> set.seed(773)
> x=rt(100,50)
> LGnormtest(x,plot=TRUE)
$rho
[1] 0.9838698

$cval
       5%
0.9871572

$pval
[1] 0.0176
```

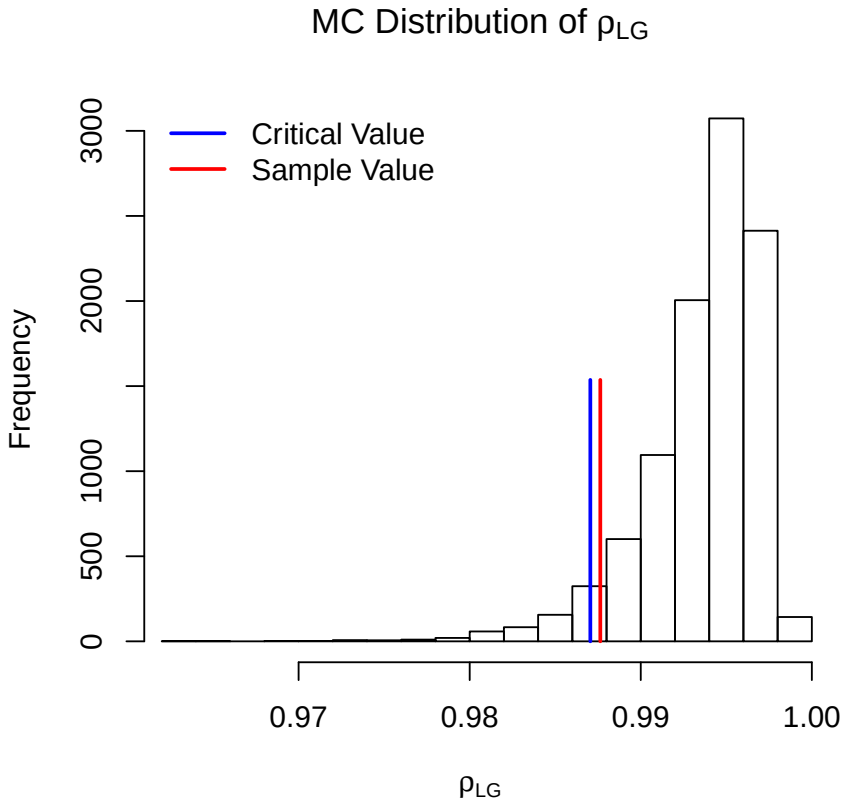


LG Test: Example #6 (Essentially-Normal Data)

```
> set.seed(773)
> x=rt(100,200)
> LGnormtest(x,plot=TRUE)
$rho
[1] 0.9876347

$cval
       5%
0.9870514

$pval
[1] 0.0601
```



SW Test: Definition

Shapiro-Wilk (SW) normality test uses the test statistic

$$W = \frac{(\sum_{i=1}^{n} a_i y_{(i)})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where the a_i coefficients are defined through the relation

$$\mathbf{a}' = (a_1, \ldots, a_n) = \frac{\mathbf{m}'\mathbf{V}^{-1}}{[\mathbf{m}'\mathbf{V}^{-2}\mathbf{m}]^{1/2}}$$

with $\mathbf{m} = \{m_i\}_{n\times 1}$ containing the expected values of the $y_{(i)}$ statistics and $\mathbf{V} = \{v_{ij}\}_{n\times n}$ denoting the covariance matrix of the $y_{(i)}$ statistics under the assumption that $Y \sim \mathrm{N}(\mu, \sigma^2)$.

Note: $y_{(i)}$ are the order statistics corresponding to y_i for $i \in \{1, \ldots, n\}$.

SW Test: Definition (continued)

Test is $H_0 : Y \sim \mathrm{N}(\mu, \sigma^2)$ versus $H_1 : Y \not\sim \mathrm{N}(\mu, \sigma^2)$.

Note that $\sum_{i=1}^{n} a_i y_{(i)}$ is the best linear unbiased estimate (BLUE) of the regression slope predicting the $y_{(i)}$ scores from the m_i values.

- Remember $R^2 = SSR/SST$ with $SSR = \hat{b}_1^2 \sum_{i=1}^{n} (x_i - \bar{x})^2$ and $SST = \sum_{i=1}^{n} (y_i - \bar{y})^2$
- So W test statistic is related to R^2 in model: $y_{(i)} = b_0 + b_1 m_i + e_i$
- Reject H_0 if observed W is too small

Shapiro, S. S., & Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika, 52*, 591–611.

Royston, P. (1982) An extension of Shapiro and Wilk's W test for normality to large samples. *Applied Statistics, 31*, 115–124.

SW Test: R Function

Use `shapiro.test` in R to perform SW test on a sample of data.

Function takes in a data vector, and returns the W test statistic and (approximate) p-value of $H_0 : Y \sim \mathrm{N}(\mu, \sigma^2)$ versus $H_1 : Y \not\sim \mathrm{N}(\mu, \sigma^2)$.

Function uses Royston's (1982) approximation to calculate the a_i coefficients, so the solution is approximate.

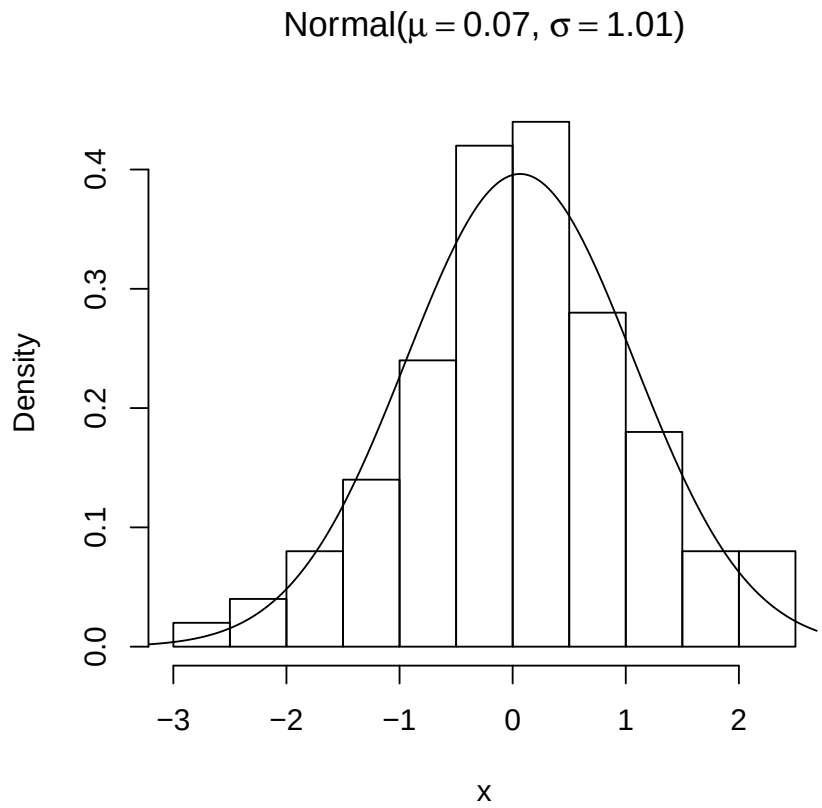
SW Test: Example #1 (Normal Data)

```
> set.seed(773)
> x=rnorm(100)
> shapiro.test(x)

 Shapiro-Wilk normality test

data:  x
W = 0.9901, p-value = 0.6713

> hnorm(x)
```



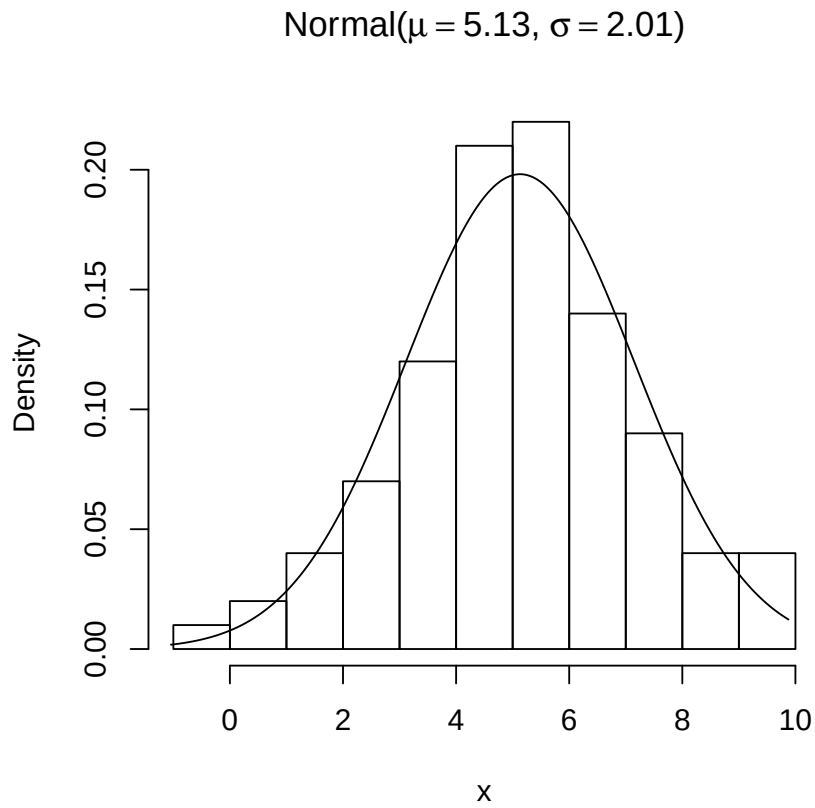
SW Test: Example #2 (Normal Data)

```
> set.seed(773)
> x=rnorm(100,mean=5,sd=2)
> shapiro.test(x)

 Shapiro-Wilk normality test

data:  x
W = 0.9901, p-value = 0.6713

> hnorm(x)
```



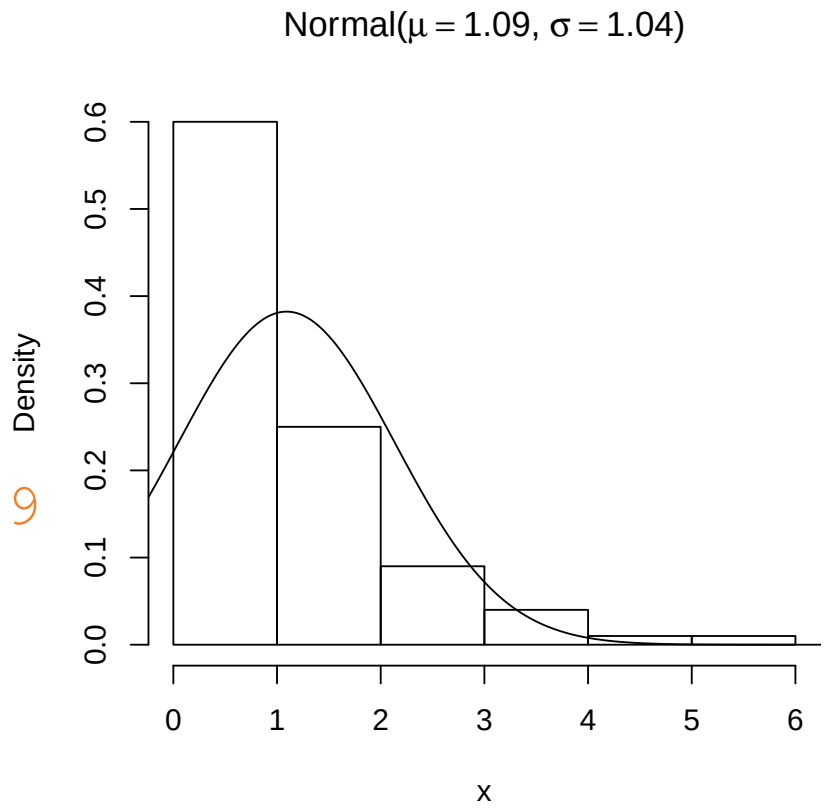
SW Test: Example #3 (Non-Normal Data)

```
> set.seed(773)
> x=rexp(100)
> shapiro.test(x)

 Shapiro-Wilk normality test

data:  x
W = 0.827, p-value = 1.823e-09

> hnorm(x)
```



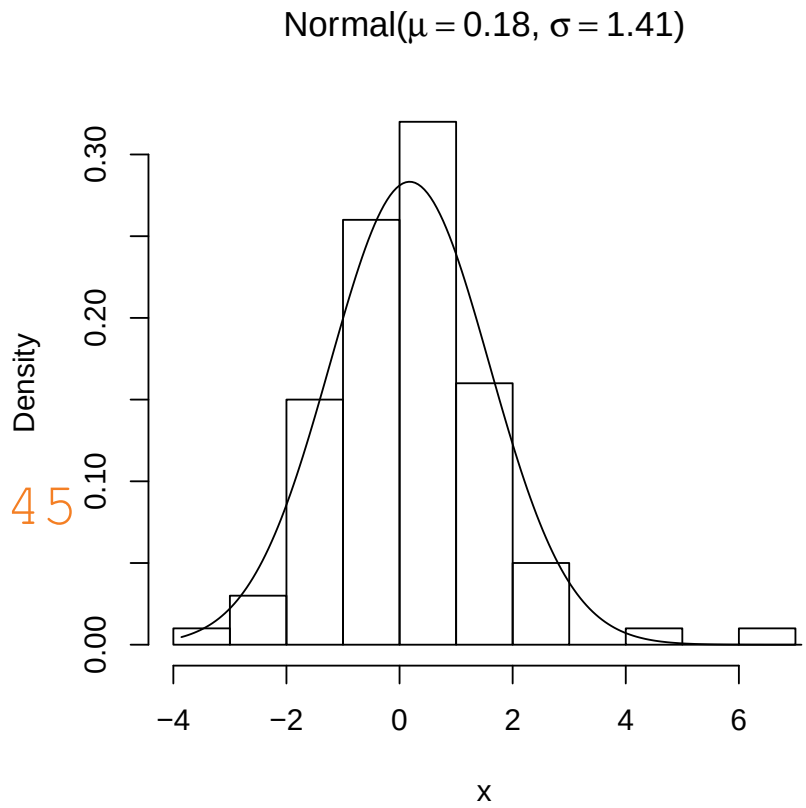
SW Test: Example #4 (Non-Normal Data)

```
> set.seed(773)
> x=rt(100,3)
> shapiro.test(x)

 Shapiro-Wilk normality test

data:  x
W = 0.9501, p-value = 0.0008445

> hnorm(x)
```



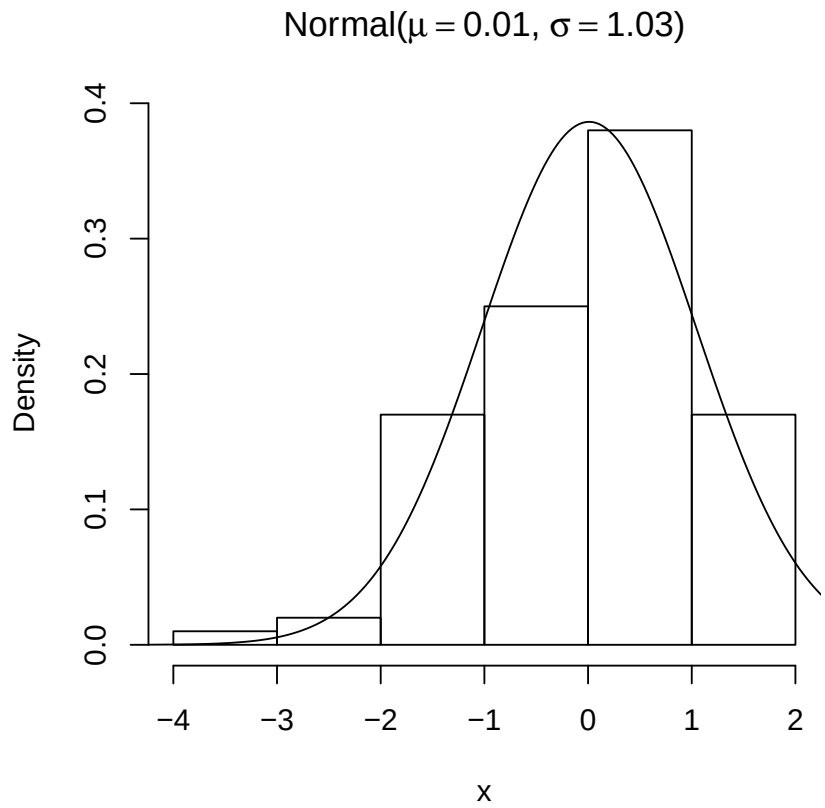
SW Test: Example #5 (Almost-Normal Data)

```
> set.seed(773)
> x=rt(100,50)
> shapiro.test(x)

 Shapiro-Wilk normality test

data:  x
W = 0.9688, p-value = 0.01797

> hnorm(x)
```



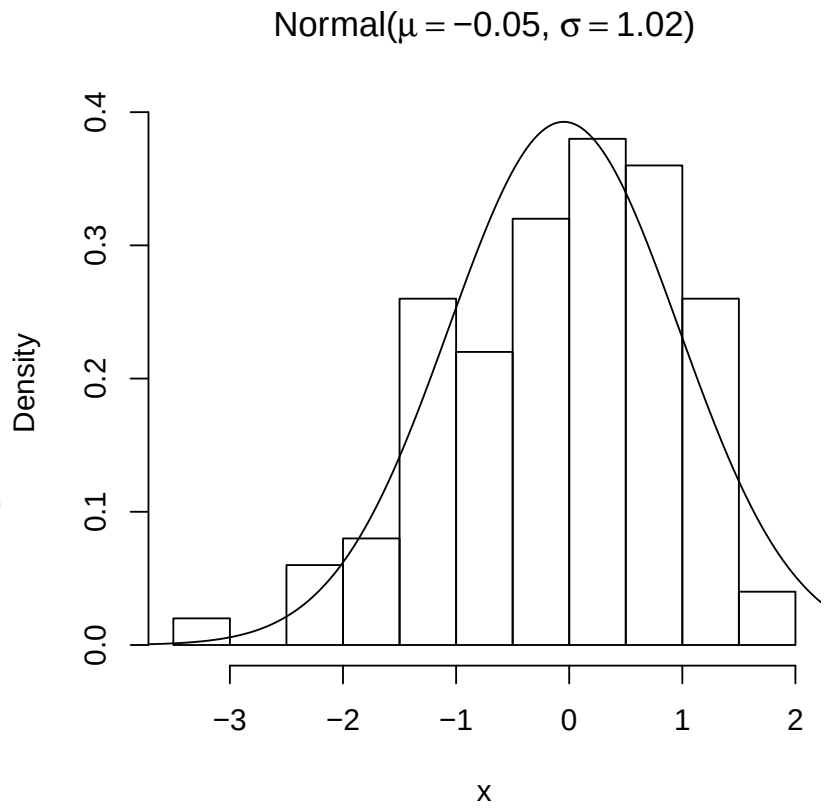
SW Test: Example #6 (Essentially-Normal Data)

```
> set.seed(773)
> x=rt(100,200)
> shapiro.test(x)

 Shapiro-Wilk normality test

data:  x
W = 0.9752, p-value = 0.05609

> hnorm(x)
```



Solutions for Non-Normality: Overview

Many possible solutions to deal with non-normality in regression:

- Bootstrap to get SE estimates for regression coefficients
- Least-squares with rank transformed data (see below reference)
- Use generalized linear model (if data is exponential family)
- Nonparametric regression

Conover, W. J., & Iman, R. L. (1981). Rank transformations as a bridge between parametric and nonparametric statistics. *The American Statistician, 35*, 124–129.

Monte Carlo Bootstrap: Definition

Suppose $x_i \overset{\text{iid}}{\sim} f(x)$ for $i \in \{1, \ldots, n\}$ from some unknown distribution f, and we want to make inferences about the statistic $T = g(x_1, \ldots, x_n)$.

Can use Monte Carlo Bootstrap:

1. Sample z_i with replacement from $\{x_1, \ldots, x_n\}$ for $i \in \{1, \ldots, n\}$
2. Calculate $T_b = g(z_1, \ldots, z_n)$ for b-th sample
3. Repeat 1–2 a total of B times to get bootstrap distribution of T
4. Compare $T = g(x_1, \ldots, x_n)$ to bootstrap distribution

Standard error of T is standard deviation of bootstrap distribution:

$$\hat{\sigma}_T = \sqrt{\frac{1}{B-1} \sum_{b=1}^{B} (T_b - \bar{T})^2}$$

where $\bar{T} = \frac{1}{B} \sum_{b=1}^{B} T_b$ is the mean of the bootstrap distribution of T.

Monte Carlo Bootstrap: Definition (continued)

Suppose we have a multiple regression model

$$y_i = b_0 + \sum_{j=1}^{p} b_j x_{ij} + e_i$$

with $e_i \overset{\text{iid}}{\sim} f(x)$ for $i \in \{1, \ldots, n\}$ where $\mathrm{E}(e_i) = 0$ and $\mathrm{E}(e_i^2) = \sigma^2$.

Can use following bootstrap procedure:

1. Fit MLR model to obtain $\hat{\mathbf{y}}$ and $\hat{\mathbf{e}} = \mathbf{y} - \hat{\mathbf{y}}$
2. Sample e_i^* with replacement from $\{\hat{e}_1, \ldots, \hat{e}_n\}$ for $i \in \{1, \ldots, n\}$
3. Define $y_i^* = \hat{y}_i + e_i^*$ and $\hat{\mathbf{b}}^* = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}^*$
4. Repeat 2–3 a total of B times to get bootstrap distribution of $\mathbf{b}$

Monte Carlo Bootstrap: R Functions

We can design our own bootstrap sampling functions:

```
bootsamp<-function(x,nsamp=10000){
  x=as.matrix(x)
  nx=dim(x)[1]
  bsamp=replicate(nsamp,x[sample.int(nx,replace=TRUE),])
}
```

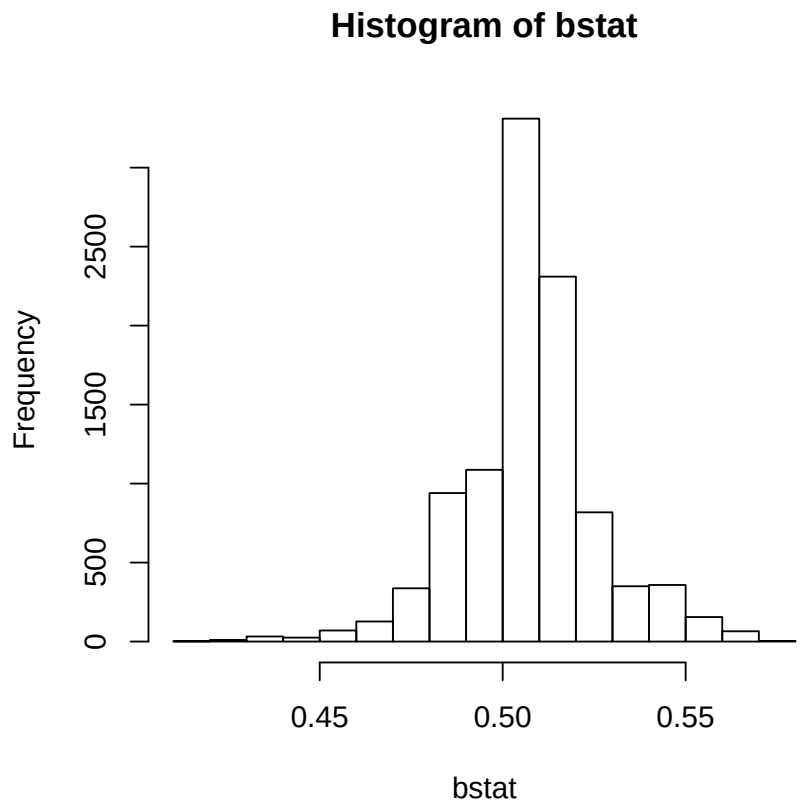
If `x` is a vector of length n, then `bootsamp` returns an $n \times B$ matrix, where B is the number of bootstrap samples (controlled via `nsamp`).

If `x` is a matrix of order $n \times p$, then `bootsamp` returns an $n \times p \times B$ array, where B is the number of bootstrap samples.

Monte Carlo Bootstrap: Examples #1 (Median)

Form a 95% confidence interval around the median of a sample:

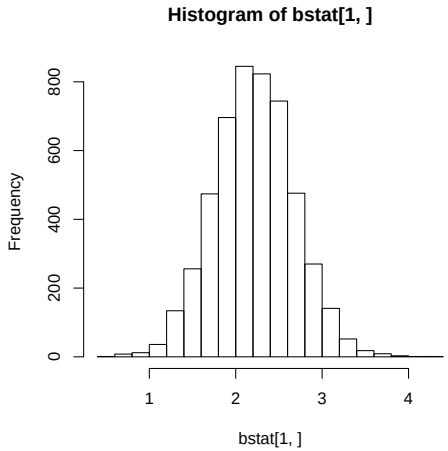
```
> set.seed(1234)
> x=runif(500)
> bsamp=bootsamp(x)
> dim(bsamp)
[1]   500 10000
> bstat=apply(bsamp,2,median)
> length(bstat)
[1] 10000
> hist(bstat)
> sd(bstat)
[1] 0.01851551
> quantile(bstat,c(.025,.975))
     2.5%     97.5%
0.4698510 0.5474821
```

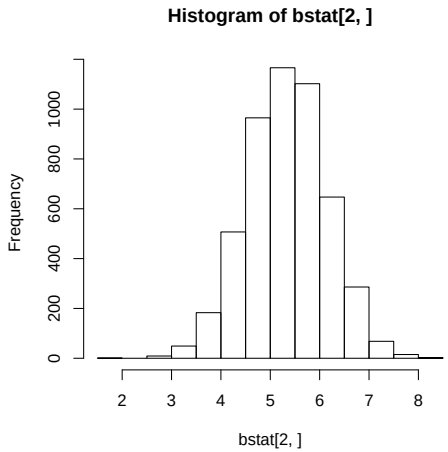


Monte Carlo Bootstrap: Examples #1 (Regression)

Form a 95% confidence interval around regression coefficients:

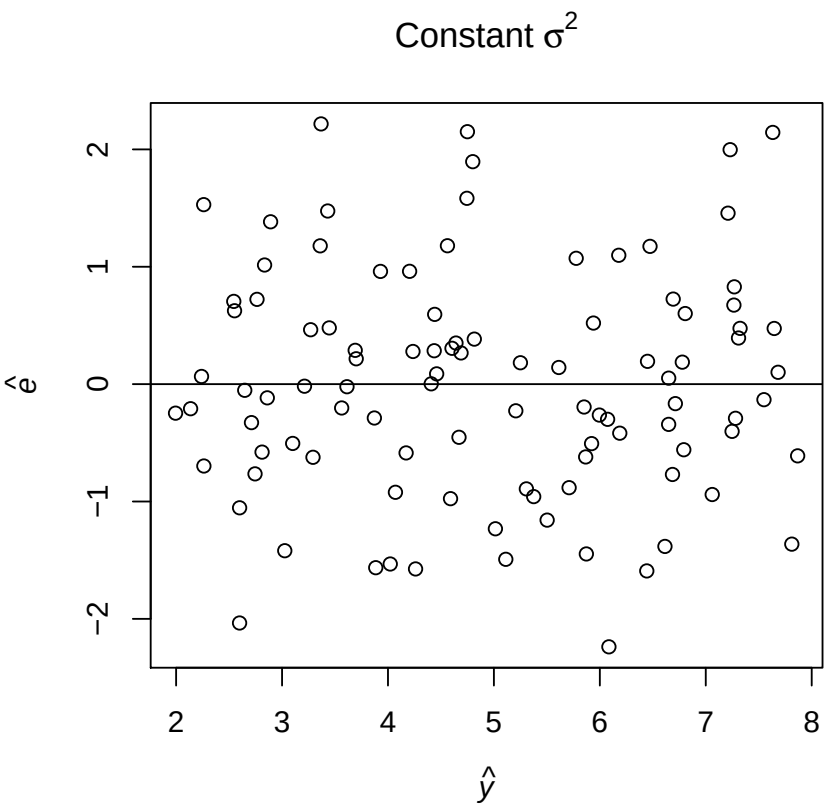
```
> set.seed(1234)
> nx=500
> x=runif(nx)
> e=c(-2*rt(nx/2,df=2),2*rt(nx/2,df=2))
> y=2.2+5*x+e
> mymod=lm(y~x)
> bsamp=bootsamp(ecent,nsamp=5000)
> bsamp=matrix(mymod$fitted,nx,5000)+bsamp
> lmcoef=function(y,x){lm(y~x)$coef}
> bstat=apply(bsamp,2,lmcoef,x=x)
> dim(bstat)
[1]    2 5000
> seval=apply(bstat,1,sd)
> cival=t(apply(bstat,1,quantile,c(.025,.975)))
> cival
                2.5%    97.5%
(Intercept) 1.313121 3.127851
x           3.777183 6.886680
```

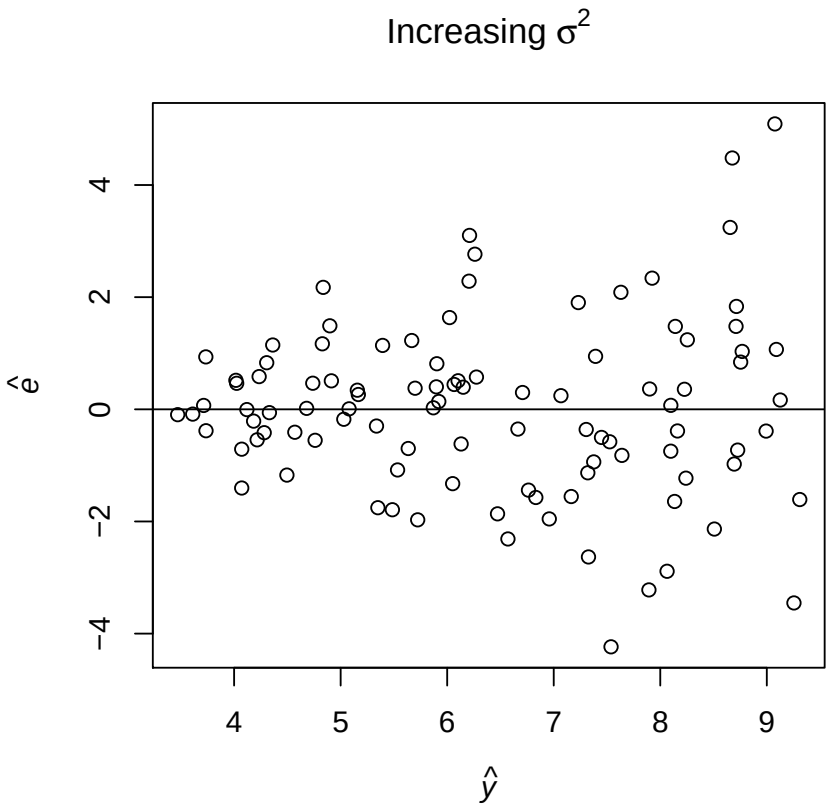




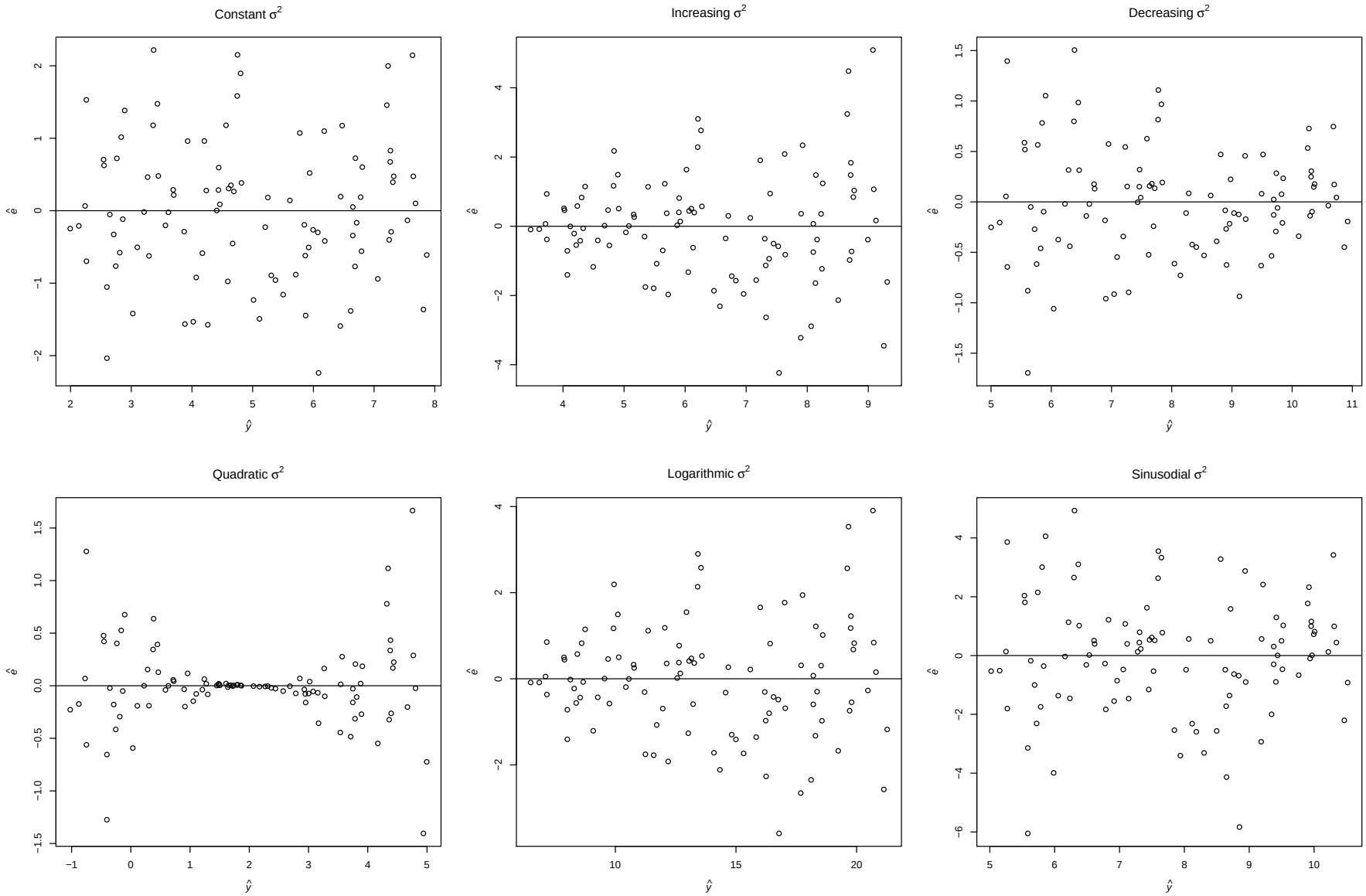
Residual Plots

To visualize the constant variance assumption, plot $\hat{y}_i$ versus $\hat{e}_i$:





Different Residual Trends



Breusch-Pagan Test: Definition

Assume a multiple regression model of the form

$$y_i = b_0 + \sum_{j=1}^{p} b_j x_{ij} + e_i$$

with $e_i \overset{\text{iid}}{\sim} f(x)$ for $i \in \{1, \ldots, n\}$ where $\mathrm{E}(e_i) = 0$ and $\mathrm{E}(e_i^2) = \sigma^2$.

Consider the auxiliary model predicting the squared error terms

$$e_i^2 = \gamma_0 + \sum_{j=1}^{p} \gamma_j x_{ij} + \tilde{e}_i$$

where $\boldsymbol{\gamma} = (\gamma_0, \gamma_1 \ldots, \gamma_p)'$ are the auxiliary coefficients, and $\tilde{\mathbf{e}} = (\tilde{e}_i, \ldots, \tilde{e}_n)'$ is the corresponding auxiliary error vector.

Breusch-Pagan Test: Definition (continued)

To test $H_0 : V(e_i) = \sigma^2$ versus $H_1 : V(e_i) \neq \sigma^2$ use

$$\chi^2_{BP} = n\tilde{R}^2$$

where $\tilde{R}^2$ is coefficient of multiple determination from auxiliary model.

As $n \to \infty$, we have $\chi^2_{BP} \to \chi^2_p$; so reject H_0 if $\chi^2_{BP} > \chi^2_{p(\alpha)}$.

Note: asymptotic (large sample) test of heteroskedasticity.

Breusch-Pagan Test: R Function

We can design our own Breusch-Pagan test function:

```
BPtest=function(mymod){
  mymod$model[,1]=(mymod$resid)^2
  newmod=lm(formula(mymod),data=mymod$model)
  modsum=summary(newmod)
  Rsq=modsum$r.squared
  BPstat=Rsq*(dim(mymod$model)[1])
  pval=1-pchisq(BPstat,modsum$df[1]-1)
  list(BP=BPstat,df=modsum$df[1]-1,pval=pval)
}
```

Note: function input is an object created by `lm` function.

Breusch-Pagan Test: Examples

```
> set.seed(123)
> x=runif(100)*2
> y=2+3*x+rnorm(100)
> linmod=lm(y~x)
> BPtest(linmod)
$BP
[1] 0.02521226

$df
[1] 1

$pval
[1] 0.8738393
```

```
> set.seed(123)
> x=(0.5+runif(100)*2)
> y=2+3*x+rnorm(100,sd=x)
> linmod=lm(y~x)
> BPtest(linmod)
$BP
[1] 13.984

$df
[1] 1

$pval
[1] 0.0001843734
```

Non-Constant Error Variance: Overview

MLR model assumes that $e_i \overset{\text{iid}}{\sim} \text{N}(0, \sigma^2)$, which includes the assumption of homogeneity of variance.

If $\text{E}(e_i) = \sigma_i^2$, then we have heteroskedasticity.

We have two possible approaches:

- Correct our parameter estimates via weighted least squares
- Correct our standard error estimates via sandwich SE estimates

Weighted Least Squares: Overview

Assuming $\mathbf{e} \sim \mathrm{N}(\mathbf{0}, \mathbf{W}^{-1})$ with $\mathbf{W} = \mathrm{diag}(1/\sigma_1^2, \ldots, 1/\sigma_n^2)$, we have

$$L(\mathbf{b}|\mathbf{y}, \mathbf{X}, \mathbf{W}) = (2\pi)^{-n/2}(\prod_{i=1}^{n} \sigma_i^2)^{1/2} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{Xb})'\mathbf{W}(\mathbf{x}-\mathbf{Xb})}$$

due to the assumptions of the MLR model.

If $\mathbf{W}$ is known, the MLE (or WLS) regression coefficients are

$$\hat{\mathbf{b}}_w = (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}\mathbf{X}'\mathbf{W}\mathbf{y}$$

and the covariance matrix of $\hat{\mathbf{b}}_w$ is given by $\sigma^2_{\hat{\mathbf{b}}_w} = \sigma^2(\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}$.

Weighted Least Squares: Derivation

Think of WLS as using OLS on transformed variables.

If we premultiply $\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{e}$ by the matrix $\mathbf{W}^{1/2}$ we have

$$\tilde{\mathbf{y}} = \tilde{\mathbf{X}}\mathbf{b} + \tilde{\mathbf{e}}$$

where $\tilde{\mathbf{y}} = \mathbf{W}^{1/2}\mathbf{y}$, $\tilde{\mathbf{X}} = \mathbf{W}^{1/2}\mathbf{X}$, and $\tilde{\mathbf{e}} = \mathbf{W}^{1/2}\mathbf{e}$.

Note that $\mathrm{V}(\tilde{\mathbf{e}}\tilde{\mathbf{e}}') = \mathbf{W}^{1/2}\mathrm{V}(\mathbf{e}\mathbf{e}')\mathbf{W}^{1/2} = \mathbf{W}^{1/2}\mathbf{W}^{-1}\mathbf{W}^{1/2} = \mathbf{I}_n$ and

$$\begin{aligned}\hat{\mathbf{b}}_w &= (\tilde{\mathbf{X}}'\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}'\tilde{\mathbf{y}} \\ &= (\mathbf{X}'\mathbf{W}\mathbf{X})^{-1}\mathbf{X}'\mathbf{W}\mathbf{y}\end{aligned}$$

is the OLS solution using reweighted entries.

Weighted Least Squares: In Practice

In practice we never know $\mathbf{W}$, so we have to estimate the weights.

Considering the auxiliary model predicting the squared error terms

$$e_i^2 = \gamma_0 + \sum_{j=1}^{p} \gamma_j x_{ij} + \tilde{e}_i$$

we could define $\hat{w}_i = (\hat{\gamma}_0 + \sum_{j=1}^{p} \hat{\gamma}_j x_{ij})^{-1}$ and iteratively update weights and regression coefficients until convergence (see Faraway book).

Weights could be estimated using different functions of the residuals:

Holland, P. W., & Welsch, R. E. (1977). Robust regression using iteratively reweighted least-squares. *Communications in Statistics: Theory and Methods, 6*, 813–827.

Weighted Least Squares: R Function

In R, you can use the `rlm` (robust linear models) function from the `MASS` (Modern Applied Statistics with S) package.

The `rlm` function works much like the `lm` function, but uses iteratively reweighted least-squares (IRWLS).

Can specify many different weighting functions; default is Huber, which defines the weights via the absolute value of the scaled residuals.

Weighted Least Squares: Example

```
> library(MASS)
> set.seed(123)
> x=rnorm(100)
> y=2+3*x+rnorm(100,sd=x^2)
> mydata=data.frame(y=y,x=x)
> olsmod=lm(y~x,mydata)
> olsmod$coef
(Intercept)           x
   1.797853    2.899911
> irwlsmod=rlm(y~x,mydata)
> irwlsmod$coef
(Intercept)           x
    1.92587     2.96380
> irwlsmod$w[1:4]
[1] 1.0000000 1.0000000 0.9003846 1.0000000
```

Sandwich Standard Errors: Overview

Remember that the covariance matrix of $\hat{\mathbf{b}}$ is given by

$$\mathrm{V}(\hat{\mathbf{b}}) = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathrm{V}(\mathbf{y})\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathrm{E}(\mathbf{e}\mathbf{e}')\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$$

which simplifies to $\mathrm{V}(\hat{\mathbf{b}}) = \sigma^2(\mathbf{X}'\mathbf{X})^{-1}$ under the assumption that $e_i \overset{\text{iid}}{\sim} \mathrm{N}(0, \sigma^2) \iff \mathbf{e} \sim \mathrm{N}(\mathbf{0}_n, \sigma^2\mathbf{I}_n)$.

Instead of assuming that $\mathbf{e} \sim \mathrm{N}(\mathbf{0}_n, \sigma^2\mathbf{I}_n)$, we could define

$$\mathrm{V}_S(\hat{\mathbf{b}}) = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\hat{\mathrm{E}}(\mathbf{e}\mathbf{e}')\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}$$

where the covariance matrix $\hat{\mathrm{E}}(\mathbf{e}\mathbf{e}')$ is estimated from the residuals.

Sandwich Standard Errors: R Function

A few different options for getting sandwich SE estimates in R:

- `hccm` function (`car` package)
- `Rsandcov` function (`haplo.ccs` package)
- `vcovHC` function (`sandwich` package)

We will use `hccm` (Heteroscedasticity-Corrected Covariance Matrices), which can estimate $\hat{\mathrm{E}}(\mathbf{e}\mathbf{e}')$ using a variety of methods.

Using `type="hc0"` defines $\hat{\mathrm{E}}(\mathbf{e}\mathbf{e}') = \mathrm{diag}(\hat{e}_1^2, \ldots, \hat{e}_n^2)$, which is the classic correction proposed by White (1980).

Sandwich Standard Errors: Example

```
> library(car)
> set.seed(123)
> x=rnorm(100)
> y=2+3*x+rnorm(100,sd=x^2)
> olsmod=lm(y~x)
> summary(olsmod)$coef
             Estimate Std. Error  t value     Pr(>|t|)
(Intercept) 1.797853  0.1176536 15.28090 1.110520e-27
x           2.899911  0.1289032 22.49681 1.731134e-40
> sand=hccm(olsmod,type="hc0")
> sqrt(diag(sand))
(Intercept)           x
  0.1083337   0.2383926
> X=cbind(1,x); colnames(X)<-c("(Intercept)","x")
> XtXi=solve(crossprod(X))
> cmat=tcrossprod(XtXi,X)%*%diag(olsmod$resid^2)%*%X%*%XtXi
> sqrt(diag(cmat))
(Intercept)           x
  0.1083337   0.2383926
```

Influence in Regression: Overview

Remember the hat matrix is defined as $\mathbf{H} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$, and we know that $\hat{\mathbf{e}} \sim \mathrm{N}(\mathbf{0}_n, \sigma^2(\mathbf{I}_n - \mathbf{H})) \iff \mathrm{V}(\hat{e}_i) = \sigma^2(1 - h_{ii})$.

The diagonal elements of the hat matrix h_{ii} are the *leverage* values.

Note that $\hat{y}_i = \sum_{j=1}^{n} h_{ij} y_j$, so an observation with a large leverage value has a (potentially) large influence on the solution.

- Large h_{ii} results from extreme predictor variable scores.

Rule of thumb: leverages larger than $2\bar{h}$ should be looked at more closely, where $\bar{h} = \frac{1}{n}\sum_{i=1}^{n} h_{ii}$ is the mean leverage.

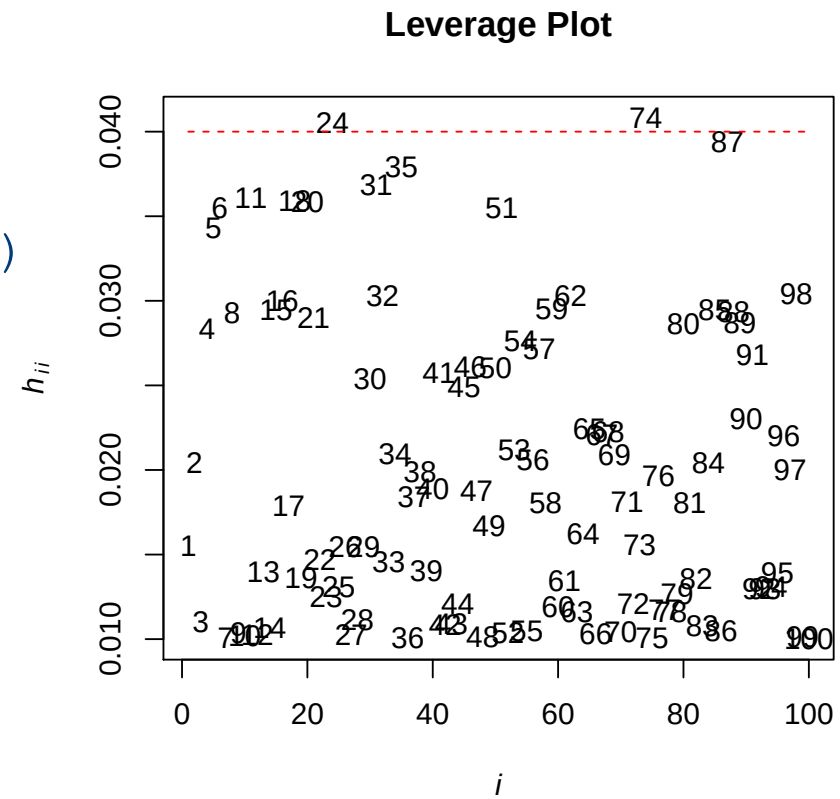
Leverage Plots

To visualize the equal influence assumption, we can define our own leverage plot function:

```
levplot<-function(mymod,k=2,ptext=TRUE,...){
  nx=dim(mymod$model)[1]
  hii=influence(mymod)$hat
  hbar=mean(hii)
  if(ptext){
    plot(1:nx,hii,type="n",xlab=expression(italic(i)),
         ylab=expression(italic(h[ii])),main="Leverage Plot")
    text(1:nx,hii,1:nx)
  } else{plot(1:nx,hii,xlab=expression(italic(i)),
              ylab=expression(italic(h[ii])),
              main="Leverage Plot")}
  lines(c(1,nx),c(k*hbar,k*hbar),...)
}
```

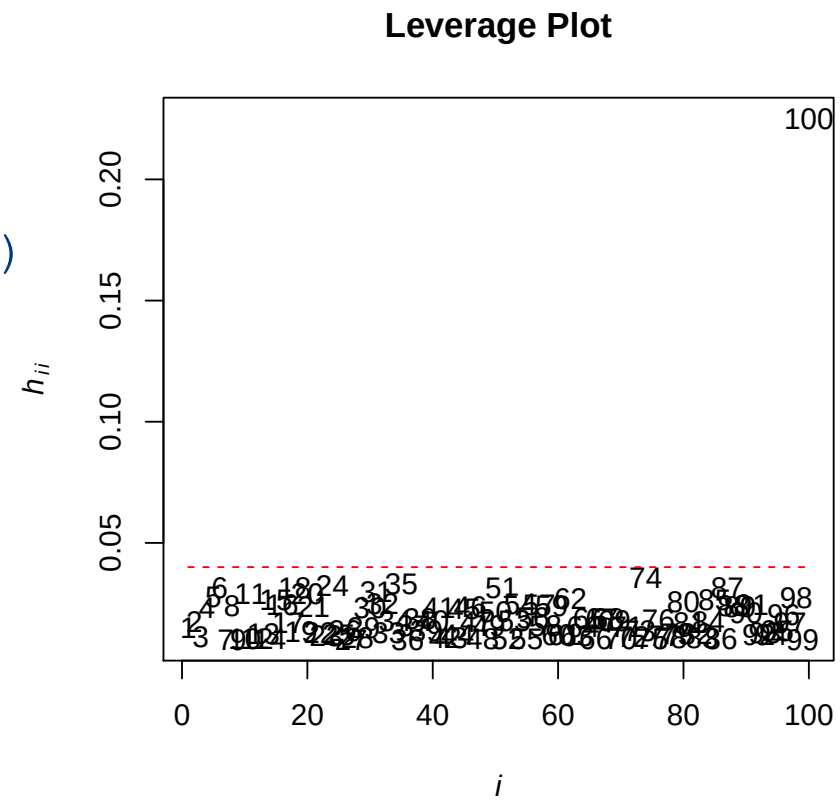
Leverage Plots: Example #1 (Equal Influence)

```
> set.seed(123)
> x=runif(100)
> y=2+3*x+rnorm(100)
> mymod=lm(y~x)
> levplot(mymod,col="red",lty=2)
```



Leverage Plots: Example #2 (Unequal Influence)

```
> set.seed(123)
> x=c(runif(99),2)
> y=2+3*x+rnorm(100)
> mymod=lm(y~x)
> levplot(mymod,col="red",lty=2)
```



Cook's Distance: Definition

To test for unequal influence, we can use *Cook's distance*.

Let $\hat{\mathbf{b}}_{(i)}$ denote the OLS estimate of $\mathbf{b}$ holding out the i-th observation, and let $\hat{\mathbf{y}}_{(i)} = \mathbf{X}\hat{\mathbf{b}}_{(i)}$ denote the corresponding fitted values.

Now note that $\hat{\mathbf{y}}_{(i)} - \hat{\mathbf{y}} = \mathbf{X}(\hat{\mathbf{b}}_{(i)} - \hat{\mathbf{b}})$, which implies that

$$(\hat{\mathbf{y}}_{(i)} - \hat{\mathbf{y}})'(\hat{\mathbf{y}}_{(i)} - \hat{\mathbf{y}}) = (\hat{\mathbf{b}}_{(i)} - \hat{\mathbf{b}})'\mathbf{X}'\mathbf{X}(\hat{\mathbf{b}}_{(i)} - \hat{\mathbf{b}})$$

Cook's Distance: Definition (continued)

Cook's (1977) distance D_i is defined as

$$D_i = \frac{(\hat{\mathbf{b}}_{(i)} - \hat{\mathbf{b}})'\mathbf{X}'\mathbf{X}(\hat{\mathbf{b}}_{(i)} - \hat{\mathbf{b}})}{(p+1)\hat{\sigma}^2}$$

$$= \frac{\hat{e}_i^2}{(p+1)\hat{\sigma}^2}\left[\frac{h_{ii}}{(1+h_{ii})^2}\right]$$

where

- $\hat{\mathbf{b}}$ is OLS estimate of $\mathbf{b}$ with all observations included
- $\hat{\mathbf{b}}_{(i)}$ is OLS estimate of $\mathbf{b}$ holding out the i-th observation
- $(p+1)$ is the number of columns of $\mathbf{X}$
- $\hat{\sigma}^2$ is the MSE with all observations included
- $\hat{e}_i$ is the estimated residual with all observations included
- h_{ii} is the leverage value for the i-th observation

Cook's Distance: Definition (continued)

Remembering that the $100(1-\alpha)\%$ confidence ellipsoid for $\mathbf{b}$ is the set of vector $\mathbf{b}^*$ staisfying

$$\frac{(\mathbf{b}^* - \hat{\mathbf{b}})'\mathbf{X}'\mathbf{X}(\mathbf{b}^* - \hat{\mathbf{b}})}{(p+1)\hat{\sigma}^2} \leq F^{(\alpha)}_{p+1,n-p-1}$$

we see that D_i approximates an $F_{p+1,n-p-1}$ distribution.

Use $F^{(1-\alpha)}_{p+1,n-p-1}$ critical values to determine if observation is an outlier.

- Note if $D_i \approx F^{(0.5)}_{p+1,n-p-1}$, then holding out i-th observation moves OLS estimate to edge of 50% confidence region
- Typically want $\hat{\mathbf{b}}_{(i)}$ to say within 5–10% (or less) region.

Cook's Distance: R Functions

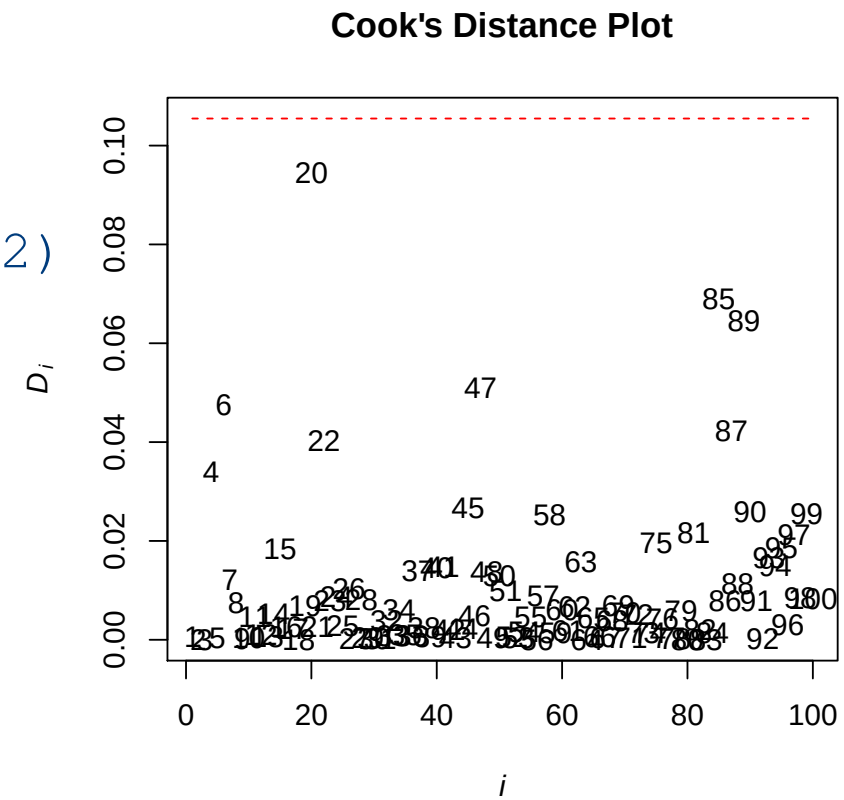
Get estimates of Cook's distance using `cooks.distance` function.

We can design our own Cook's distance plotting function:

```
cookplot<-function(mymod,k=NULL,alpha=0.1,ptext=TRUE,...){
  nx=dim(mymod$model)[1]
  np=length(mymod$coef)
  cdist=cooks.distance(mymod)
  if(is.null(k)){k=qf(alpha,np,nx-np)}
  ylim=range(cdist)
  if(ylim[1]>k){ylim[1]=k} else if(ylim[2]<k){ylim[2]=k}
  if(ptext){
    plot(1:nx,cdist,type="n",xlab=expression(italic(i)),ylim=ylim,
         ylab=expression(italic(D[i])),main="Cook's Distance Plot")
    text(1:nx,cdist,1:nx)
  } else{plot(1:nx,cdist,xlab=expression(italic(i)),ylim=ylim,
              ylab=expression(italic(D[i])),main="Cook's Distance Plot")}
  lines(c(1,nx),c(k,k),...)
}
```

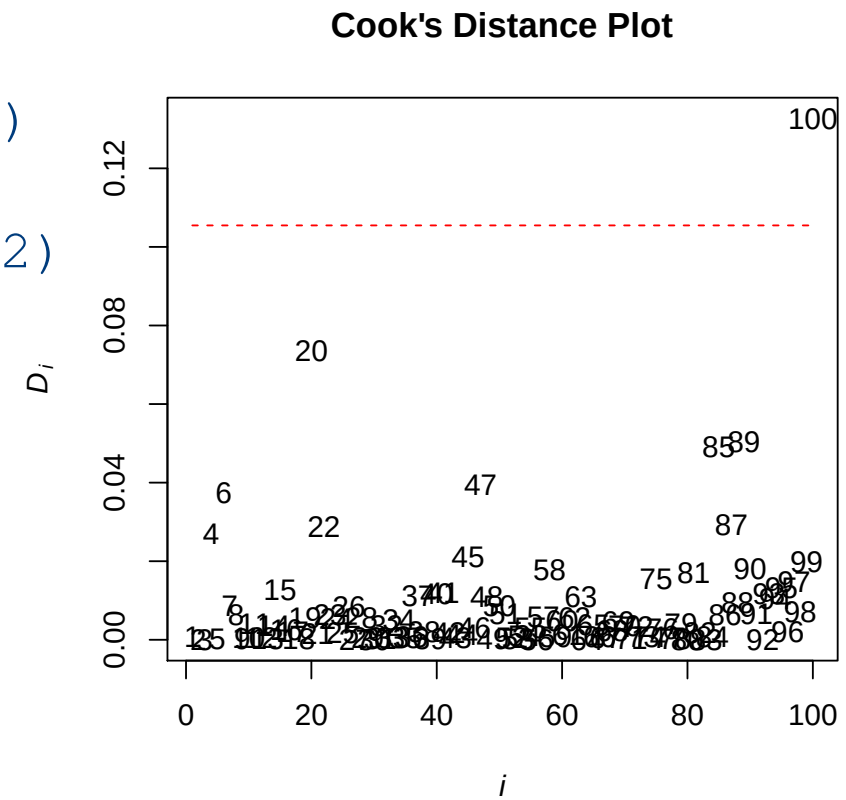
Cook's Distance: Examples #1 (Equal Influence)

```
> set.seed(123)
> x=runif(100)
> y=2+3*x+rnorm(100)
> mymod=lm(y~x)
> cookplot(mymod,col="red",lty=2)
```



Cook's Distance: Example #2 (Unequal Influence)

```
> set.seed(123)
> x=runif(100)
> y=c(2+3*x[1:99],-1)+rnorm(100)
> mymod=lm(y~x)
> cookplot(mymod,col="red",lty=2)
```



Solutions for Non-Equal Influence

Many possible solutions to deal with non-equal influence:

- Rank (or other) transformation of data
- IRWLS (`rlm` function in `MASS` package)
- Regression trees (`cv.tree` function in `tree` package)
- Minimize L_1 norm (`lqnorm` function in `VGAM` package)
- Quantile regression (`rq` function in `quantreg` package)

We will discuss data transformations in Notes 6 Part B; feel free to examine the other options if interested.