# R-Code from Mahler's Guides to **CAS Exam MAS-2**

prepared by
Howard C. Mahler, FCAS
Copyright ©2026 by Howard C. Mahler.

Howard Mahler
hmahler@mac.com
www.howardmahler.com/Teaching

For many examples in my MAS-2 Study Guides, I provide corresponding R-Code.
The order presented here is the same as that of my Study Guides.
(Not every Study Guide has any R-Code.)

You will not be asked to write R-Code on your exam.

Some of you will benefit from running R-Code, either mine or that in the textbooks, in order to get a better understanding of how the various algorithms operate.

I have tried to keep my R-Code simple.
I have not tried to produce the most elegant or efficient code.

Please let me know of any suspected errors.
hmahler@mac.com

# NonParametric Credibility

*My R-Code for my Workers Compensation Pure Premium Example:*

```
install.packages("insuranceData")  # you only need to install a package once
library(insuranceData)
data(WorkersComp)
attach(WorkersComp)
head(WorkersComp)

WorkersComp$PR<-WorkersComp$PR/100  # converting payroll to units of $100
train=WorkersComp[YR < 6,]  # years 1 to 5
test=WorkersComp[YR > 5,]  # years 6 and 7

trainPayroll<-matrix(train$PR,c(5,121))  # payroll for training dataset arranged in 5 rows
mi<-colSums(trainPayroll)  # payrolls by class
m<-sum(mi)  # total payroll
Pi<-m - sum(mi^2)/m  # denominator of the VHM
Pi

trainLoss<-matrix(train$LOSS,c(5,121))  # losses for training dataset arranged in 5 rows
Xi<-trainLoss/trainPayroll  # pure premium by class and year
XiBar<-colSums(trainLoss)/mi  # pure premium by class
XiBar[1:5]
XBar<-sum(train$LOSS)/sum(train$PR)  # the overall pure premium for the training dataset
XBar

temp<-matrix(1:(5*121),c(5,121))
temp[1,]<-temp[2,]<-temp[3,]<-temp[4,]<-temp[5,]<-XiBar
# created a matrix in which each row has the class pure premiums

EPV<-sum(trainPayroll*(temp-Xi)^2, na.rm = TRUE)  # numerator of the EPV
EPV
EPV<-EPV/(4*121)
EPV

VHM<-sum((XiBar - XBar)^2 * mi) - (121-1)*EPV   # numerator of the VHM
VHM
VHM<-VHM/Pi
VHM

K<-EPV/VHM
K
Zi<-mi/(mi+K)
Zi[1:5]
```

```
XBarPrime=sum(Zi * XiBar)/sum(Zi)  # the credibility weighted pure premium
XBarPrime

predict<-Zi*XiBar + (1-Zi)*XBarPrime
predict[1:5]

testPayroll<-matrix(test$PR,c(2,121))  # payroll for test dataset arranged in 2 rows
testLoss<-matrix(test$LOSS,c(2,121))  # losses for test dataset arranged in 2 rows
testXi<-testLoss/testPayroll
testmi<-colSums(testPayroll)
testXiBar<-colSums(testLoss)/testmi  #pure premiums for test dataset by class
testXiBar[1:5]

plot(testXiBar, predict, cex=0.5, xlab="Observed PP Years 6 & 7",
        ylab="Predicted PP", main="Using the Credibility Weighted PP")
abline(c(0,1), lwd=2, col="blue")

plot(testXiBar[Zi>0.5], predict[Zi>0.5], cex=0.5, xlab="Observed PP Years 6 & 7",
        ylab="Predicted PP",
        main="Using the Credibility Weighted PP; Credibility Greater Than 50%")
abline(c(0,1) ,lwd=2, col="blue")
```

*My R-Code for my Workers Compensation Frequency Example:*

```
# the data is on the website of Predictive Modeling Applications in Actuarial Science
# https://instruction.bus.wisc.edu/jfrees/jfreesbooks/PredictiveModelingVol1/index.htm
# you are interested in Chapter 16 of Volume 1
# you need to download the data from their website onto your computer
# then exactly how you get the data into R depends on you operating system
# how I read the data into R on my Mac after downloading the data from their website
WCFreq<-read.table(file.choose(), header = TRUE)
# you need to point to the location on your computer of the file WorkersFreqData.txt

attach(WCFreq)
head(WCFreq)
train<-WCFreq[WCFreq$yearcentred<1,]  # training data is the first four years
test<-WCFreq[WCFreq$yearcentred>0,]  # test data excludes the first four years
test<-test[test$yearcentred<3,]  # test data excludes the seventh year

train<-train[train$riskclass!=4,]
train<-train[train$riskclass!=54,]
train<-train[train$riskclass!=61,]
train<-train[train$riskclass!=86,]  # Eliminating four classes with missing years of data
dim(train)
test<-test[test$riskclass!=4,]
test<-test[test$riskclass!=54,]
test<-test[test$riskclass!=61,]
test<-test[test$riskclass!=86,]  # Eliminating four classes with missing years of data
dim(test)

trainPayroll<-matrix(train$payroll, c(4,126))
mi<-colSums(trainPayroll)
mi[1:5]
m<-sum(mi)
m
Pi<-m - sum(mi^2)/m  # denominator of the VHM
Pi

trainCounts<-matrix(train$count, c(4,126))
Xi<-trainCounts/trainPayroll  # frequency by class and year
XiBar<-colSums(trainCounts)/mi  # frequency by class
XBar<-sum(trainCounts)/sum(trainPayroll)  # the overall frequency for the training dataset
XBar

temp<-matrix(1:(4*126),c(4,126))
temp[1,]<-temp[2,]<-temp[3,]<-temp[4,]<-XiBar
# created a matrix in which each row has the class frequencies
```

```
EPV<-sum(trainPayroll*(temp-Xi)^2, na.rm = TRUE)  # numerator of the EPV
EPV
EPV<-EPV/(3*126)
EPV

VHM=sum((XiBar - XBar)^2 * mi) - 125*EPV  # numerator of the VHM
VHM
VHM<-VHM/Pi
VHM

K<-EPV/VHM
K
Zi<-mi/(mi+K)
Zi[1:5]
summary(Zi)

XBarPrime=sum(Zi * XiBar)/sum(Zi)  # the credibility weighted frequency
XBarPrime
predict<-Zi*XiBar + (1-Zi)*XBarPrime
predict[1:5]

testPayroll<-matrix(test$payroll, c(2,126))
testCounts<-matrix(test$count, c(2,126))
testXi<-testCounts/testPayroll
testmi<-colSums(testPayroll)
testXiBar<-colSums(testCounts)/testmi
testXiBar[1:5]

plot(testXiBar,predict, cex=0.5, xlab="Observed Freq. Years 5 & 6", ylab="Predicted Frequency")
abline(c(0,1), lwd=2, col="blue")

plot(testXiBar[Zi>0.8], predict[Zi>0.8], cex=0.5,xlab="Observed Freq. Years 5 & 6",
        ylab="Predicted Frequency", main="Credibility Greater Than 80%")
abline(c(0,1), lwd=2, col="blue")
```

# Advanced Statistical Learning

## Section 4, Classification Trees

```
install.packages("tree")
library(tree)
library(datasets)
data(iris)

iris.tree=tree(Species~.,data=iris)
summary(iris.tree)
plot(iris.tree)
text(iris.tree , pretty=0)

cv.iris=cv.tree(iris.tree)
plot(cv.iris$size, cv.iris$dev , type="b")

prune.iristree=prune.misclass(iris.tree , best=4)
summary(prune.iristree)
plot(prune.iristree)
text(prune.iristree , pretty=0)
```

From Solutions to Problems:

```
install.packages("mlbench")
library(mlbench)
data(PimaIndiansDiabetes2)
install.packages("tree")
library(tree)
diabetes.tree=tree(diabetes~., data=PimaIndiansDiabetes2)
diabetes.tree=prune.misclass(diabetes.tree , best=9)
plot(diabetes.tree)
text(diabetes.tree , pretty=0, cex=0.9)
```

## Section 5, Bagging

```
install.packages("tree")
library(tree)
install.packages("randomForest")
library(randomForest)

dos=c(1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58, 61, 64, 67, 70)
eff=c(3 , 5 ,10 , 9, 10, 11, 16, 18, 15, 44, 62, 85, 95, 90, 82, 59, 50, 28, 15, 7, 11, 4, 12, 3)

drug.tree=tree(eff~dos)
plot(drug.tree)
text(drug.tree,pretty=0)

s1=sample(1:24,replace=TRUE)
s1
        9 8 12 21 8 14 17 10 20 22 11 7 18 21 9 12 21 10 18 17 21 1 18 12

drug.tree1=tree(eff[s1]~dos[s1])
plot(drug.tree1)
text(drug.tree1,pretty=0)

bag.drug=randomForest(eff~dos,mtry=1,importance=TRUE)
bag.drug
        Call:
        randomForest(formula = eff ~ dos, mtry = 1, importance = TRUE)
        Type of random forest: regression
        Number of trees: 500
        No. of variables tried at each split: 1
        Mean of squared residuals: 76.12459
        % Var explained: 91.83

yhat.drug=predict(bag.drug,dos)
plot(dos,yhat.drug)
```

## Section 7, Boosting

```
install.packages("gbm")
library("gbm")
dos=c(1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58, 61, 64, 67, 70)
eff=c(3 , 5 ,10 , 9, 10, 11, 16, 18, 15, 44, 62, 85, 95, 90, 82, 59, 50, 28, 15, 7, 11, 4, 12, 3)

boost.drug =gbm(eff~dos, distribution="gaussian", n.trees =500 , interaction.depth =1,
        n.minobsinnode =1, shrinkage=0.01)
yhat.boostdrug=predict(boost.drug)
plot(dos,yhat.boostdrug)
```

*My R Code to Apply Boosting to the Heart Disease Data:*

```
install.packages("gbm") # only need to install a package once
library(gbm)

install.packages("kmed") # only need to install a package once
library(kmed)

attach(heart)
heart$sex <- as.integer(as.logical(heart$sex)) # convert True/False to 1/0
heart$fbs <- as.integer(as.logical(heart$fbs))
heart$exang <- as.integer(as.logical(heart$exang))
heart$class<-+ sapply(heart$class, as.logical) # convert values greater than 1 to 1

set.seed (1)
train <- sample (1: nrow(heart), nrow(heart) / 2)
boost.heart =gbm(class~.,data=heart[train,], distribution="gaussian", n.trees = 5000,
        interaction.depth = 2, shrinkage = 0.01)

for(i in 1:10)
        print((mean (( predict(boost.heart , newdata = heart[train,], n.trees = 100 *i) -
        heart$class[train])^2))) # mean squared training errors

for(i in 1:10)
        print((mean (( predict(boost.heart , newdata = heart[-train,], n.trees = 100 *i) -
        heart$class[-train])^2))) # mean squared test errors
```

## Section 8, BART

```
install.packages("BART") # only need to install a package once
library(BART)

install.packages("kmed") # only need to install a package once
library(kmed)

attach(heart)
heart$sex <- as.integer(as.logical(heart$sex)) # convert True/False to 1/0
heart$fbs <- as.integer(as.logical(heart$fbs))
heart$exang <- as.integer(as.logical(heart$exang))
heart$class<-+ sapply(heart$class, as.logical) # convert values greater than 1 to 1

set.seed (1)
train <- sample (1: nrow(heart), nrow(heart) / 2)

xheart<-heart[,1:13]
yheart<-heart[,14]
xhearttrain=xheart[train,]
yhearttrain=yheart[train]
xhearttest=xheart[-train,]
yhearttest=yheart[-train]

bartheartfit<-pbart(xhearttrain,yhearttrain,x.test=xhearttest)

mean((yhearttest - bartheartfit$prob.test.mean)^2)
```

## Section 9, Principal Components Analysis

```
x=matrix(c(-2, 1, 3, 6, 11, 15, 17, 20, 1, -1, 4,-4, 0, 8, -8, -6, -4, 0, 4, 8, 12, 16, 20, 24) , ncol=3)
agents=prcomp(x,scale=TRUE)

agents$rotation
agents$x

agents$sdev^2

agents$sdev^2/sum(agents$sdev^2)

biplot(agents,scale=0)
```

Iris Data Example:

```
library(datasets)
data(iris)
iris.meas=iris[,c(1,2,3,4)]
names(iris.meas)
pca.out=prcomp(iris.meas,scale=TRUE)
pca.out$rotation
pca.out$sdev^2
pca.out$sdev^2/sum(pca.out$sdev^2)
```

From Solutions to Problems:

```
library(ISLR)
data(Auto)
mtcars.pca<-prcomp(mtcars[,c(1:7,10,11)],center=TRUE,scale=TRUE)
biplot(mtcars.pca,scale=0,xlabs=1:32)
mtcars.pca$rotation
mtcars.pca$x
summary(mtcars.pca)
```

## *Section 10, Missing Values and Matrix Completion*

```
qqna=matrix(c(8,11,18,16,21,25,27,18,11,9,14,11,10,18,11,4,24,24,14,18, 22,26,30, 34),ncol=3)
qindex.na=matrix(c(3,8,4,7,1,2,1,1,2,2,3,3),ncol=2)
qqna[qindex.na]<-NA

getmissing<-function(X){
return(createindices(X)[which(is.na(X)),])}

getnonmissing<-function(X){
return(createindices(X)[which(!is.na(X)),])}

createindices<-function(X){
nrows<-dim(X)[1]
ncols<-dim(X)[2]
cols<-rep(NA,nrows*ncols)
for(i in 1:ncols){cols[(1:nrows)+(i-1)*nrows]<-i}
rows<-rep(1:nrows,ncols)
return(cbind(rows,cols))}

pcapprox<-function(pc,means){
c<-createindices(pc$x)
return(means[c[,2]]+pc$x[c[,1],1]*pc$rotation[c[,2],1])} # Using only 1st principal component

matrixcomplete<-function(X){
d<-RMSold<-1e12
iter<-1
miss<-getmissing(X)
nonmiss<-getnonmissing(X)
Xp<-X
Xbar<-colMeans(X,na.rm=TRUE)
Xp[miss]<-Xbar[miss[,2]]
while(d>tol){
Y<-pcapprox(prcomp(Xp),Xbar)
Y<-matrix(Y, ncol=dim(X)[2])
Xp[miss]<-Y[miss]
RMS<-sum((Y[nonmiss]-Xp[nonmiss])^2)
d<-RMSold-RMS
print(c(iter, RMS))
RMSold<-RMS
iter<-iter+1
Xbar<-colMeans(Xp)}
return(Xp)}

tol<-0.001
matrixcomplete(qqna)
```

## Section 11, K-Means Clustering

```
kmeans(matrix(c(11,19,28,34,40,45,56,62,30,27,42,15,53,38),7,2),2)
```

*My R-Code for an Example on Red Wines:*

```
# one has first to download the data from
# https://archive.ics.uci.edu/dataset/186/wine+quality
# there is big blue download button
# then you have to unzip the downloaded zip file
# in the resulting folder, the data you want is winequality-red.csv

redwine <- read.csv(file.choose(), sep=";", header=TRUE)
# You need to point to where the data file has been downloaded to your computer
head(redwine, 2)

redwine.scaled<- scale(redwine)  # standardizing variables
redwine.scaled<-redwine.scaled[,2:11]  # getting rid of the variables fixed.acidity & quality
head(redwine.scaled, 2)

pcredwine<-princomp(redwine.scaled)  # principal components
pcredwine
wineloadings<-pcredwine$loadings
wineloadings[1,]
wineloadings[2,]

redwinecluster2<-kmeans(redwine.scaled, 2, nstart=30)  # K-means clustering
redwinecluster3<-kmeans(redwine.scaled, 3, nstart=30)
redwinecluster4<-kmeans(redwine.scaled, 4, nstart=30)
redwinecluster4$size

plot(pcredwine$scores[,1:2], col=redwinecluster2$cluster, pch = 2+redwinecluster2$cluster,
        xlab = "1st P.C.", ylab = "2nd P.C.", cex=0.5)
plot(pcredwine$scores[,1:2], col=redwinecluster3$cluster, pch = 1+redwinecluster3$cluster,
        xlab = "1st P.C.", ylab = "2nd P.C.", cex=0.5)
plot(pcredwine$scores[,c(1,3)], col=redwinecluster3$cluster, pch = 1+redwinecluster3$cluster,
        xlab = "1st P.C.", ylab = "3rd P.C.", cex=0.5)
plot(pcredwine$scores[,1:2], col=redwinecluster4$cluster, pch = 1+redwinecluster4$cluster,
        xlab = "1st P.C.", ylab = "2nd P.C.", cex=0.5)

redwinecluster2$tot.withinss
redwinecluster3$tot.withinss
redwinecluster4$tot.withinss

winehier<-hclust(dist(redwine.scaled))
#  hierarchical clustering, Complete linkage and Euclidean Distance are the defaults
plot(winehier)
```

From Solutions to Problems:

```
kmeans(matrix(c(48,34,79,61,25,47,82,9,33,6,7,1,84,47,50,71,45,96,67,26),10,2),3)
```

```
kmeans(matrix(
c(3,7,15,22,34,40,56,61,78,92,70,83,35,6,14,80,17,26,81,2,87,8,83,96,49,72,25,47,18,67),10,3),
3, nstart=20)
```

```
kmeans(matrix(c(24,21,35,12,18,11,27,14,4,10),5,2),2,nstart=20)
```

## Section 12, Hierarchical Clustering

```
plot(hclust (dist(matrix(c(11,19,28,34,40,45,56,62,30,27,42,15,53,38),7,2)),
        method ="complete"))


plot(hclust (dist(matrix(c(11,19,28,34,40,45,56,62,30,27,42,15,53,38),7,2)), method ="single"))


plot(hclust (dist(matrix(c(11,19,28,34,40,45,56,62,30,27,42,15,53,38),7,2)), method ="average"))


plot(hclust (dist(matrix(c(11,19,28,34,40,45,56,62,30,27,42,15,53,38),7,2)), method ="centroid"))


plot(hclust (dist(matrix(c(11,19,28,34,40,45,56,62,30,27,42,15,53,38),7,2),
        method="manhattan"), method ="single"))



ex3=matrix(c(3,7,15,22,34,40,56,61,78,92,70,83,35,6,14,80,17,26,81,2,87,8,83,96,49,72,25,47,
                18,67), 10, 3)
plot(hclust (as.dist(1-cor(t(ex3))), method ="complete"))
```

*My R-Code for An Example with Dune Vegetation:*

```
install.packages("vegan")  # you only need to install a package once
library(vegan)
data(dune)
head(dune, 5)

dd<-dist(scale(dune))  # creating a distance object from the standardized data

ccom <- hclust(dd, method = "complete")  # hierarchical clustering with the Complete linkage
plot(ccom)
rect.hclust(ccom , 3)  # show three classes

plot(ccom)
rect.hclust(ccom , 5)  # show five classes

kmeans3<-kmeans(scale(dune), 3, nstart = 100)  # algorithm tries 100 different random starts
# k-means clustering with 3 clusters; standardized data
kmeans3$cluster

install.packages("dendextend")  # you only need to install a package once
library(dendextend)

colors_vector <- c("red", "blue", "green")  # 3 colors for three k-means clusters
dend <- as.dendrogram(ccom)  # creating a dendrogram object
labels_colors(dend) <- colors_vector[kmeans3$cluster]
plot(dend)  # the previous dendrogram colored by the three k-means clusters
```

From Solutions to Problems:

```
plot(hclust(dist(c(14, 9, 43, 39, 27))^2, method="centroid"))



dat=matrix(c(5,4,3,5,5,3,2,5,6,0,3,1,6,5),7,2)
plot(hclust(dist(dat), method ="complete"))
plot(hclust(dist(dat), method ="average"))



genes=matrix(c(0.597, 0.357, 0.191, 0.693, 0.318, 0.014, 0.165, 0.975, 0.334, 0.115, 0.753,
        0.217,0.944, 0.964, 0.99, 0.894, 0.794, 0.977,0.935, 0.043, 0.451, 0.402, 0.62, 0.2), 6, 4)
plot(hclust(as.dist(1-cor(t(genes)))))



plot(hclust(as.dist(
  matrix(c(0,9,3,6,11,9,0,7,5,10,3,7,0,12,2,6,5,12,0,8,11,10,2,8,0),5,5)), method="complete"))
plot(hclust(as.dist(
  matrix(c(0,9,3,6,11,9,0,7,5,10,3,7,0,12,2,6,5,12,0,8,11,10,2,8,0),5,5)), method="single"))



data = matrix(c(0.725, 0.581, 0.846, 0.217, -0.027, -0.381, 2.442, -0.315, 1.162,   -0.252, -1.130,
        1.446, 2.033, -0.334, 1.151, -0.248, 1.938, -0.276, 0.795, 2.021),10,2)
plot( hclust(dist(data), method="complete") )

kmeans(data, 2)



plot(hclust(dist(matrix(c(13,3,20,16,8,11,15,22),4,2))^2, method="centroid"))



library(stats)
plot(hclust(dist(c(13,36,55,64)),method="single"))
plot(hclust(dist(c(13,36,55,64)),method="complete")
```

## Section 13, Neural Networks, Introduction

*My R-Code for My Simulation Example:*

```
install.packages("nnet")  # you only have to install a package once
# this package is only for neural nets with one hidden layer
# this package only uses the sigmoid activation function
library(nnet)

# simulating the data
set.seed(1776)
x=runif(200, 500, 5000)
y=log(x) + sin(x/675) + rnorm(200, 0, 0.2)
mean(y)
var(y)^0.5
plot(x, y, cex= 0.5)

# fitting the neural net to the standardized data
set.seed(1066)
nn1<-nnet(scale(x), scale(y), size=3, linout=TRUE, maxit=500)
set.seed(1492)
nn2<-nnet(scale(x), scale(y), size=3, linout=TRUE, maxit=500)
set.seed(1914)
nn3<-nnet(scale(x), scale(y), size=3, linout=TRUE, maxit=500)
set.seed(1939)
nn4<-nnet(scale(x), scale(y), size=3, linout=TRUE, maxit=500)
set.seed(1951)
nn5<-nnet(scale(x), scale(y), size=3, linout=TRUE, maxit=500)

# doing the fit five times and picking the one with the smallest squared error
# want to get a global minimum rather than a local minimum

nn2$wts  # printing out the weights for the winning model

predictions<-var(y)^.5*predict(nn2)+mean(y)  # putting the predictions on the original scale
predictions[1]
y[1]

plot(y, predictions, cex=0.5)
abline(c(0,1), lwd=2, col="blue")

plot(x, predictions, cex=0.5)
curve(log(x)+sin(x/675), add=TRUE, col="blue", lwd=2)
```

## Section 17, Recurrent Neural Networks

*Financial Series Example, My R-Code:*

```
library(ISLR2)
logvolume<-NYSE[,4]
voltrain<-logvolume[1 : 4281]
voltest<-logvolume[4282 : 6051]
arima(voltrain, order=c(5,0,0))
volforecast= -0.0125 + 0.5745*voltest[5:1769] - 0.0278*voltest[4:1768] + 0.0960*voltest[3:1767]
                     + 0.0990*voltest[2:1766] + 0.0976*voltest[1:1765]
RSS = sum((voltest[6:1770]-volforecast)^2)
RSS
TSS = sum((voltest[6:1770]-mean(voltest[6:1770]))^2)
TSS
1 - RSS/TSS
```

# Advanced GLMs

*An Example of a Lorentz Curves and the Gini Index in R:*

```
Expos<-c(267, 731, 669, 623, 259, 192, 641, 780, 725, 648, 219, 323, 224, 781, 821, 626, 205,
        866, 182, 829, 458, 801, 125, 677, 218, 812, 279, 752, 601, 684)

PP<-c(82, 66, 74, 181, 104, 174, 88, 145, 164, 53, 83, 90, 97, 187, 94, 66, 68, 57, 96, 188, 94,
        56, 193, 93, 107, 89, 178, 53, 91, 84)

ModelPP<-c(83, 26, 66, 180, 105, 136, 94, 168, 164, 25, 106, 62, 97, 213, 103, 78, 57, 57, 119,
        171, 94, 44, 182, 90, 101, 86, 181, 38, 136, 91)

Losses<-PP*Expos  # observed losses by class
sorted_Losses<-Losses[order(ModelPP)]  # sort observed losses by modeled PP
sorted_Expos<-Expos[order(ModelPP)]  # sort exposures by modeled PP

cum_Losses <- cumsum(sorted_Losses) / sum(sorted_Losses)  # percent of cumulative losses
cum_Expos<- cumsum(sorted_Expos) / sum(sorted_Expos)  # percent of cumulative exposures

plot(c(0,cum_Expos), c(0,cum_Losses), type = "l", col = "red", lwd = 2,
        xlab = "Cumulative % of Exposures", ylab = "Cumulative % of Losses")
                                # plotting the Lorentz Curve

abline(0, 1, col = "blue", lwd = 2)  # plotting the comparison line

sum(cum_Losses*sorted_Expos)/sum(sorted_Expos)
                                # computing the area under the Lorentz Curve
                                # a weighted average of heights with exposures as the weights
1 - 2*sum(cum_Losses* sorted_Expos)/sum(sorted_Expos)  # computing the Gini Index

PerfectModelPP<-PP

Losses<-PP*Expos  # observed losses by class
sorted_Losses2<-Losses[order(PerfectModelPP)]  # sort observed losses by modeled PP
sorted_Expos2<-Expos[order(PerfectModelPP)]  # sort exposures by modeled PP

cum_Losses2 <- cumsum(sorted_Losses2) / sum(sorted_Losses2)
                                 # percent of cumulative losses
cum_Expos2<- cumsum(sorted_Expos2) / sum(sorted_Expos2)
                                # percent of cumulative exposures
```

```
plot(c(0,cum_Expos2), c(0,cum_Losses2), type = "l", col = "red", lwd = 2,
        xlab = "Cumulative % of Exposures", ylab = "Cumulative % of Losses")
                                # plotting the Lorentz Curve

abline(0, 1, col = "blue", lwd = 2)  # plotting the comparison line

sum(cum_Losses2*sorted_Expos2)/sum(sorted_Expos2)
                                # computing the area under the Lorentz Curve
                                # a weighted average of heights with exposures as the weights
1 - 2*sum(cum_Losses2* sorted_Expos2)/sum(sorted_Expos2)  # computing the Gini Index
```

# Time Series

*My R-Code for the Earnings Example:*

```
install.packages("datasets")  # you only have to install a package once
library(datasets)
data(JohnsonJohnson)
earnings=JohnsonJohnson

plot(earnings, type="p", cex=0.7)

logearnings<-log(earnings)
plot(logearnings, type="p", cex=0.7)

plot(decompose(logearnings))

rand<-decompose(logearnings)$random
rand<-na.omit(rand)

acf(rand)

shapiro.test(rand)
```

Solutions to Problems Section 3:

```
acf(c(1,6,4,4,3,3,8,4,7,3),type="covariance")$acf[1]
acf(c(1,6,4,4,3,3,8,4,7,3),type="covariance")$acf[2]
acf(c(1,6,4,4,3,3,8,4,7,3))$acf[2]



acf(c(13,10,12,9,8,8,7,5),type=c("covariance"))$acf[3]
```

Solutions to Problems Section 6:

```
euro=c(1.32859, 1.33792, 1.29648, 1.30193, 1.29846, 1.31799, 1.30745, 1.33224, 1.33484,
1.36401, 1.34917, 1.37, 1.3629, 1.36476, 1.38293, 1.38058, 1.37385, 1.35994, 1.35467,
1.3329, 1.2907, 1.26707, 1.24745, 1.23209, 1.16116, 1.13501, 1.08337, 1.08073, 1.11689,
1.12229, 1.10084, 1.11287, 1.12385, 1.12291, 1.07366, 1.08936, 1.08592, 1.11019, 1.11194,
1.13398, 1.13089, 1.12447, 1.10609, 1.12036, 1.12029, 1.10337, 1.07863, 1.05384, 1.06292,
1.06489, 1.06949, 1.07088, 1.10611, 1.12297, 1.15287, 1.18161, 1.19125, 1.17494, 1.17453,
1.18453, 1.21987, 1.23459, 1.233, 1.22772, 1.18267, 1.16802, 1.16922, 1.15393, 1.16558,
1.14938, 1.13651, 1.13767)
length(euro)
acf(euro)
acf(diff(euro))
```

```
lcf=c(0.2778812, 0.2868593, 0.3065709, 0.2981785, 0.3053743, 0.3313713, 0.3474412,
0.3694949, 0.3790036, 0.3951549, 0.4198064, 0.4348348, 0.4561334, 0.4819388, 0.4904134,
0.5031087, 0.5242598, 0.5394849, 0.5413587, 0.5722101, 0.5746783, 0.5778086, 0.5924220,
0.6007279, 0.6042038, 0.5989903, 0.6221432, 0.6366923, 0.6320934, 0.6363584, 0.6407296)
plot(lcf)
acf(lcf)
acf(diff(lcf))
```

Solutions to Problems Section 7:

```
tips=c(2.72, 2.50, 2.52, 2.72, 2.40, 2.27, 2.64, 2.80, 2.67, 2.57, 2.40, 2.31, 2.24, 2.15, 1.93,
2.28, 2.45, 2.43, 2.36, 2.26, 2.18, 2.15, 2.11, 2.00, 1.96, 1.83, 1.93, 1.86, 1.80, 1.77, 1.96, 1.98,
1.89, 2.05, 2.11, 2.09, 2.00, 1.96, 2.13, 2.39, 2.45, 2.50, 2.47, 2.27, 2.26, 2.34, 2.19, 2.23, 2.40,
2.36, 2.25)
length(tips)
mean(tips)
plot(tips)
acf(tips)
pacf(tips)
arima(tips,order=c(1,0,0))
```

*R-Code for My Baseball Example:*

```
yank=c(0.489, 0.638, 0.463, 0.391, 0.523, 0.404, 0.527, 0.669, 0.510, 0.417, 0.500, 0.671,
0.623, 0.545, 0.546, 0.481, 0.536, 0.512, 0.424, 0.383, 0.359, 0.390, 0.355, 0.414, 0.552, 0.409,
0.286, 0.344, 0.429, 0.442, 0.386, 0.305, 0.393, 0.390, 0.403, 0.333, 0.338, 0.349, 0.298, 0.429,
0.344, 0.331, 0.364, 0.461, 0.467, 0.435, 0.370, 0.390, 0.370, 0.364, 0.364, 0.383, 0.344, 0.331,
0.377, 0.370, 0.364, 0.403, 0.487, 0.370)
length(yank)
acf(yank)
pacf(yank)
arima(yank,order=(c(3,0,0)))
arima(yank,order=(c(3,0,1)))
```

*R-Code for Runs Per Game Example:*

```
rpg<-c(4.01,4.36,4.58,4.61,4.85,4.55,4.18,4.61, 4.38,4.48,4.45,4.31,4.28,4.38,4.31,4.53,
4.46,3.95,4.04,3.99,3.99,3.77,3.42,4.07,4.34,3.89,3.69,4.21,4.12,4.21,3.99,4.47,4.1,4.46,4.29,
4,4.3,4.31,4.26,4.33,4.41,4.72,4.14,4.13,4.26,4.31,4.12,4.6,4.92,4.85,5.04,4.77, 4.79,5.08,
5.14,4.78,4.62,4.73,4.81,4.59,4.86,4.80,4.65,4.61,4.38,4.28,4.32,4.17,4.07,4.25)
length(rpg)
acf(rpg)
pacf(rpg)
arima(rpg,order = c(1, 0, 0))
arima(rpg,order = c(3, 0, 0))
arima(rpg,order = c(1, 0, 1))
```

Solutions to Problems Section 12:

```
q43<-c(112, 104, 100, 96, 101, 101, 105, 94, 106, 106, 108, 110)
arima(q43,order=c(2,0,0))
arima(q43,order=c(0,0,1))
arima(q43,order=c(2,0,1))
```

*My R-Code for the Nile River Flow Example*:

```
install.packages("datasets") # you only have to install a package once
library(datasets)
data(Nile)

plot(Nile ,type="p", cex=0.7)
acf(Nile)
pacf(Nile)

plot(diff(Nile) ,type="p", cex=0.7)
acf(diff(Nile))
pacf(diff(Nile))

arima(Nile,c(1,0,0))
arima(Nile,c(0,0,1))
arima(Nile,c(1,0,1))
arima(Nile,c(1,1,1))
arima(Nile,c(2,1,1))
arima(Nile,c(1,1,2))

nile.model<-arima(Nile,c(1,1,1))
var(Nile)

plot(resid(nile.model), type="p", cex=0.7)
acf(resid(nile.model))

qqnorm(resid(nile.model), cex=0.5)
qqline(resid(nile.model), col = "blue", lwd = 2)
```

*My R-Code for the Sales Example*:

```
install.packages("datasets") # you only have to install a package once
library(datasets)
data(BJsales)
sales<-BJsales

plot(sales,type="p",cex=0.7)
acf(sales)
pacf(sales)

plot(diff(sales),type="p",cex=0.7)
acf(diff(sales))
pacf(diff(sales))

arima(sales,c(1,0,0))
arima(sales,c(0,0,1))
arima(sales,c(1,0,1))
arima(sales,c(1,1,1))
arima(sales,c(2,1,1))
arima(sales,c(1,1,2))

sales.model<-arima(sales,c(1,1,1))
var(sales)

plot(resid(sales.model),type="p",cex=0.7)
acf(resid(sales.model))

qqnorm(resid(sales.model),cex=0.5)
qqline(resid(sales.model), col = "blue", lwd = 2)

shapiro.test(resid(sales.model))

install.packages("stats") # you only have to install a package once
library(stats)
Box.test(resid(sales.model),type="Ljung-Box",10)
```

*My R-Code for the Unemployment Example*:

```
install.packages("astsa") # you only have to install a package once
library(astsa)
data(UnempRate)

plot(UnempRate, type="p", cex=0.7)

plot(decompose(UnempRate))

rand<-decompose(UnempRate)$random
rand<-na.omit(rand)
acf(rand)

model<-arima(UnempRate, order=c(3,1,1), seas=list(order=c(2,0,2),12))
model

var(UnempRate)

plot(resid(model), type="p", cex=0.5)

acf(resid(model))
Box.test(resid(model), type="Ljung-Box",10)

qqnorm(resid(model), cex=0.5)
qqline(resid(model), col = "blue", lwd = 2)

shapiro.test(resid(model))
```

# Linear Mixed Models

For my Weight Loss Example (in my Section 2):

```
install.packages("nlme") # you only have to install a package once
library(nlme)
individual=c(1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6)
time=c(0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3)
weight=c(154,150,144,142,187,180,175,169,149,147,140,134,222,214,206,196,176,167,164,
                160,130,126,120,118)

simplemodel<-lme(weight~time, random = ~1 | individual, method = "ML")
summary(simplemodel)
random.effects(simplemodel)

# Fitting instead via Restricted Maximum Likelihood
simplemodel2<-lme(weight~time, random = ~1 | individual, method = "REML")
summary(simplemodel2)
random.effects(simplemodel2)
```

For my Claim Counts Example (in my Section 2):

```
install.packages("nlme") # you only have to install a package once
library(nlme)

prior=c(0, 0, 1, 0, 1, 0, 1, 1, 2, 3, 4, 2, 1, 3, 3, 0, 1, 4, 5, 2, 1, 0, 2, 2, 2, 3, 1, 2, 0, 1, 0, 1, 3, 1, 0,
        5, 2, 2, 6, 2, 0, 0, 1, 1, 1, 1, 2, 3, 1, 2, 4, 2, 3, 4, 0, 2, 4, 2, 1, 8, 0, 0, 2, 1, 5, 2, 3, 4, 5, 4,
        0, 2, 1, 0, 3, 2, 1, 4, 5, 3,2, 2, 1, 5, 5, 4, 1, 1, 5, 2, 2, 4, 4, 2, 5, 3, 6, 0, 5, 3, 1, 2, 0, 0, 4, 1,
        6, 1, 2, 2, 4, 2, 1, 1, 0, 6, 2, 1, 4, 2)

post=c(0, 0, 1, 0, 1, 3, 0, 0, 0, 3, 0, 0, 2, 3, 2, 2, 1, 4, 1, 3, 1, 3, 0, 1, 0, 0, 4, 0, 1, 3, 4, 0, 3, 0, 3,
        2, 2, 2, 1, 5, 3, 2, 1, 1, 2, 0, 3, 1, 2, 2, 7, 1, 1, 4, 4, 3, 2, 2, 6, 6, 1, 1, 1, 3, 0, 1, 0, 3, 3, 1,
        2, 2, 3, 4, 5, 2, 5, 5, 3, 4, 1, 2, 1, 4, 3, 2, 2, 1, 2, 1, 6, 4, 0, 2, 6, 2, 5, 3, 3, 3, 4, 2, 1, 3, 6,
        3, 0, 1, 5, 2, 5, 3, 2, 2, 3, 3, 5, 2, 7, 4)

class=c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
        3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6)

simplemodelcounts<-lme(post~prior, random = ~1 | class, method = "ML")
summary(simplemodelcounts)
random.effects(simplemodelcounts)

# Fitting instead via Restricted Maximum Likelihood
simplemodelcounts2<-lme(post~prior, random = ~1 | class, method = "REML")
summary(simplemodelcounts2)
random.effects(simplemodelcounts2)
```

For my Grain Yield Example (in my Section 2):

```
install.packages("lme4") # you only have to install a package once
library(lme4)

seed=c(0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1)

plot=c(1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4)

year=c(1,1,1,1,2,2,2,2,3,3,3,3,1,1,1,1,2,2,2,2,3,3,3,3)

yield=c(920,1084,1195,954,722,691,690,835,808,972,1002,1295,
        1232,1168,1017,1220,933,1004,907,989,762,1029,1086,1209)

simplemodelyields<-lmer(yield~seed + (1 | plot) + (1 | year), REML = F)
summary(simplemodelyields)
ranef(simplemodelyields)

# Fitting instead via Restricted Maximum Likelihood
simplemodelyields2<-lmer(yield~seed + (1 | plot) + (1 | year), REML = T)
summary(simplemodelyields2)
ranef(simplemodelyields2)
```

For the Sleep Study Example (in my Section 2):

```
install.packages("lme4") # you only have to install a package once
library(lme4)

attach(sleepstudy)
sleepstudy[1:13,]
require(lattice)
xyplot(Reaction ~ Days | Subject, sleepstudy, type = c("g","p","r"),
        index = function(x,y) coef(lm(y ~ x))[1],
        xlab = "Days of sleep deprivation",
        ylab = "Average reaction time (ms)", aspect = "xy")

fm1 <- lmer(Reaction ~ Days + (Days|Subject), sleepstudy, subset=Days>=2)
summary(fm1)
ranef(fm1)

install.packages("merTools") # you only have to install a package once
library(merTools)
plotREsim(REsim(fm1)[1:18,])
plotREsim(REsim(fm1)[19:36,])

# By default, lme4 assumes that all coefficients associated with the same random-effects term
        are correlated.
# A model with instead independent random effects
fm2 <- lmer(Reaction ~ Days + (1|Subject) + (0+Days|Subject), sleepstudy, subset=Days>=2)
summary(fm2)
ranef(fm2)
```

For the EM Algorithm Example (in my Section 5):

```
install.packages("mixtools")  # only need to install a package once
library(mixtools)

ESL=c(-0.39,0.12,0.94,1.67,1.76,2.44,3.72,4.28,4.92,5.53,0.06,0.48,1.01,1.68,1.80,3.25,4.12,
        4.60,5.28,6.22)

hist(ESL)

normalmixEM(ESL, lambda=0.5, mu=c(1,4), sigma=c(1,1))
# Algorithm converged in 16 iterations
```

For My Revised Simple Weight Loss Example (in my Section 9):

```
install.packages("nlme") # you only have to install a package once
library(nlme)
individual=c(1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6)
time=c(0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3)
weight2=c(164,160,154,152,177,170,165,159,169,167,160,154,172,164,156,146,176,167,164,
          160,170,166,160,158)
simplemodel3<-lme(weight2~time, random = ~1 | individual, method = "REML")
summary(simplemodel3)
random.effects(simplemodel3)
predict(simplemodel3)
```

For the Stroke Example (in my Section 12):

```
install.packages("lme4") # you only have to install a package once
library(lme4)

scores=c(45,45,45,45,80,80,80,90,20,25,25,25,30,35,30,50,50,50,55,70,70,75,90,90,25,25,35,
40,60,60,70,80,100,100,100,100,100,100,100,100,20,20,30,50,50,60,85,95,30,35,35,40,50,60,
75,85,30,35,45,50,55,65,65,70,40,55,60,70,80,85,90,90,65,65,70,70,80,80,80,80,30,30,40,45,
65,85,85,85,25,35,35,35,40,45,45,45,45,45,80,80,80,80,80,80,15,15,10,10,10,20,20,20,35,35,
35,45,45,45,50,50,40,40,40,55,55,55,60,65,20,20,30,30,30,30,30,30,35,35,35,40,40,40,40,40,
35,35,35,40,40,40,45,45,45,65,65,65,80,85,95,100,45,65,70,90,90,95,95,100,25,30,30,35,40,40
,40,40,25,25,30,30,30,30,35,40,15,35,35,35,40,50,65,65)

time<-rep(seq.int(1,8),24)

group = c(rep(0,64),rep(1,64),rep(2,64))
group.f<-factor(group)

individual=c(rep(1,8),rep(2,8),rep(3,8),rep(4,8),rep(5,8),rep(6,8),rep(7,8),rep(8,8),rep(9,8),
rep(10,8),rep(11,8),rep(12,8),rep(13,8),rep(14,8),rep(15,8),rep(16,8),rep(17,8),rep(18,8),
rep(19,8),rep(20,8),rep(21,8),rep(22,8),rep(23,8),rep(24,8))

model1<-lmer(scores~time+group.f+time:group.f + (time|individual))
summary(model1)

model2<-lmer(scores~time+time:group.f + (time|individual))
summary(model2)

install.packages("nlme") # you only have to install a package once
library(nlme)

random.effects(model2)

install.packages("merTools") # you only have to install a package once
library(merTools)

REsim(model2)
plotREsim(REsim(model2)[1:24,])
plotREsim(REsim(model2)[25:48,])
```

For the Severity Example (in my Section 12):

```
install.packages("lme4") # you only have to install a package once
library(lme4)

state=c(1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,
        4,4,5,5,5,5,5,5,5,5,5,5,5,5)
period=c(1,2,3,4,5,6,7,8,9,10,11,12,1,2,3,4,5,6,7,8,9,10,11,12,1,2,3,4,5,6,7,8,9,10,11,12,1,2,3,4,
        5,6,7,8,9,10,11,12,1,2,3,4,5,6,7,8,9,10,11,12)
severity=c(1738, 1642, 1794, 2051, 2079, 2234, 2032, 2035, 2115, 2262, 2267, 2517, 1364,
        1408, 1597, 1444, 1342, 1675, 1470, 1448, 1464, 1831, 1612, 1471, 1759, 1685,
        1479, 1763, 1674, 2103, 1502, 1622, 1828, 2155, 2233, 2059, 1233, 1146, 1010,
        1257, 1426, 1532, 1953, 1123, 1343, 1243, 1762, 1306, 1456, 1499, 1609, 1741,
        1482, 1572, 1606, 1735, 1607, 1573, 1613, 1690)
claims=c(7861, 9251, 8706, 8575, 7917, 8263, 9456, 8003, 7365, 7832, 7849, 9077, 1622,
        1742, 1523, 1515, 1622, 1602, 1964, 1515, 1527, 1748, 1654, 1861, 1147, 1357,
        1329, 1204, 998, 1077, 1277, 1218, 896, 1003, 1108, 1121, 407, 396, 348, 341,
        315, 328, 352, 331, 287, 384, 321, 342, 2902, 3172, 3046, 3068, 2693, 2910,
        3275, 2697, 2663, 3017, 3242, 3425)

lmmHach.ind<- lmer(severity ~ period + (1 | state) +  (0 + period | state), weights=claims/1000)
# independent random slopes and intercepts
summary(lmmHach.ind)
ranef(lmmHach.ind)

lmmHach2 <- lmer(severity ~ period+(1|state),weights=claims/1000)
# only random intercepts
summary(lmmHach2)
anova(lmmHach.ind, lmmHach2)
```

My R-Code for the Mexican Insurance Example (in mySection 14):

```
# the data is on the website of Predictive Modeling Applications in Actuarial Science
# https://instruction.bus.wisc.edu/jfrees/jfreesbooks/PredictiveModelingVol1/index.htm
# you are interested in Chapter 14 of Volume 1
# you need to download the data from their website onto your computer
# then exactly how you get the data into R depends on you operating system
# how I read the data into R on my Mac after downloading the data from their website
mxsectors = read.csv(file.choose())
# you need to point to the location on your computer of the file FES2010.csv

attach(mxsectors)
dim(mxsectors)
head(mxsectors)

states<-as.factor(mxsectors$ENTIDAD)
sectors<-as.factor(mxsectors$OPERACION.O.RAMO)
premiums<-mxsectors$PRIMA.EMITIDA
incurredlosses<-mxsectors$SINIESTROS.OCURRIDOS  # I will not use these
paidlosses<-mxsectors$SINIESTROS.PAGADOS

ex1<-which(premiums<=0)
ex2<-which(paidlosses<=0)
ex3<-which(incurredlosses<=0)
ex4<-which(states=="Extranjero")
exclude<-c(ex1, ex2, ex3, ex4)  # records I will exclude from the analysis

states<-states[-exclude]
sectors<-sectors[-exclude]
premiums<-premiums[-exclude]
paidlosses<-paidlosses[-exclude]

plot(log(premiums),log(paidlosses),cex=0.3,xlab="log written premiums",ylab="log paid losses")

install.packages("lme4")  # you only need to install a package once
library(lme4)
install.packages("lattice")  # you only need to install a package once
library(lattice)

sectormodel<-lmer(log(paidlosses) ~ log(premiums) + (1 | states) + (1 | sectors), REML = F)
summary(sectormodel)
randomeffects<-ranef(sectormodel, condVar = TRUE)
randomeffects

aggregate(premiums, by=list(Sectors=sectors), FUN=sum)
aggregate(premiums, by=list(States=states), FUN=sum)
```

```
p1 <- dotplot(randomeffects)[["states"]]
p1

p2 <- dotplot(randomeffects)[["sectors"]]
p2

pred<-predict(sectormodel)

plot(log(paidlosses), pred, cex=0.3, xlab="log observed paid losses", ylab="fitted")
abline(c(0,1), lwd=2, col="blue")
```

Solutions to Problems Section 14:

```
install.packages("MASS") # you need only install a package once
library(MASS)
attach(michelson)
install.packages("lme4") # you need only install a package once
library(lme4)
model.mich <- lmer(Speed ~ (1|Run) + (1|Expt), REML = T)
install.packages("merTools") # you need only install a package once
library(merTools)
REsim(model.mich)
plotREsim(REsim(model.mich))
```

For Simulating the Authors' Example 1 (in mySection 15):

```
install.packages("nlme") # you only have to install a package once
library(nlme)
simweight <- function(intercept, sigma){
        weight<-intercept+rnorm(4,0,sigma)
        return(weight) }
simweights <- function(samples, sigma, sigmalitter, diff){
        weights <- rep(NA, times = samples*8)
        for (i in 1:samples){
        # mean weight for treatment 1 is assumed to be 10
        weights[4*(i-1)+1:4]<-simweight(10+rnorm(1,0,sigmalitter),sigma)
        # mean weight for treatment 2 is assumed to be 10 + diff
        weights[4*samples +4*(i-1)+1:4]<-simweight(10 + diff + rnorm(1,0,sigmalitter),sigma) }
        return(weights) }

createlitter<-function(samples){
        lit<-rep(1,4)
        for (i in 2:(samples*2)){
        lit<-c(lit,rep(i,4))}
        return(lit)}
createtreat<-function(samples){
        t<-c(rep(1,4*samples), rep(2,4*samples))
        return(t)}
testdiff<-function(samples, sigma, sigmalitter, litter, treatment, diff){
        w<-simweights(samples, sigma, sigmalitter, diff)
        pvalue<-anova(lme(w~treatment, random = ~1 | litter, method = "ML"))[2,4]
        return(pvalue)}
testdiffs <- function(trials, samples, sigma, sigmalitter, litter, treatment, diff){
        pvalues <- rep(NA, times = trials)
        for (i in 1:trials){
        pvalues[i]<-testdiff(samples, sigma, sigmalitter, litter, treatment, diff)}
        return(pvalues)}

simpowers <- function(trials, samples, sigma, sigmalitter, diff){
        litter <-createlitter(samples)
        treatment<-createtreat(samples)
        powers<- rep(NA, times = 5)
        pp<-testdiffs(trials, samples, sigma, sigmalitter, litter, treatment, diff)
        powers[1]<-sum(pp<0.1)
        powers[2]<-sum(pp<0.05)
        powers[3]<-sum(pp<0.025)
        powers[4]<-sum(pp<0.01)
        powers[5]<-sum(pp<0.005)
        powers<-powers/trials
        return(powers) }
simpowers(10000, 10, 0.4, 0.3, 0.5)
```

For Simulating the Simple Weight Example (in my Section 15):

```
install.packages("nlme") # you only have to install a package once
library(nlme)

simweight <- function(intercept, slope, sigma){
        weight<-intercept+c(0,1,2,3)*slope+rnorm(4,0,sigma)
        return(weight) }

simweights <- function(samples, Beta0, slope, sigma, sigmaindiv){
        weights <- rep(NA, times = samples*4)
        for (i in 1:samples){
        weights[4*(i-1)+1:4]<-simweight(Beta0+rnorm(1,0,sigmaindiv),slope,sigma) }
        return(weights) }

createindiv<-function(samples){
        ind<-rep(1,4)
        for (i in 2:samples){
        ind<-c(ind,rep(i,4))}
        return(ind)}

testslope<-function(samples, Beta0, slope, sigma, sigmaindiv, individual, time){
        w<-simweights(samples,Beta0, slope, sigma, sigmaindiv)
        pvalue<-anova(lme(w~time, random = ~1 | individual, method = "ML"))[2,4]
        return(pvalue)}

testslopes <- function(trials, samples, Beta0, slope, sigma, sigmaindiv, individual, time){
        pvalues <- rep(NA, times = trials)
        for (i in 1:trials){
        pvalues[i]<-testslope(samples,Beta0, slope, sigma, sigmaindiv, individual, time)}
        return(pvalues)}

simpowers <- function(trials, samples, Beta0, slope, sigma, sigmaindiv){
        individual<-createindiv(samples)
        time<-rep(c(0,1,2,3),samples)
        powers<- rep(NA, times = 5)
        pp<-testslopes(trials,samples,Beta0, slope, sigma, sigmaindiv, individual, time)
        powers[1]<-sum(pp<0.1)
        powers[2]<-sum(pp<0.05)
        powers[3]<-sum(pp<0.025)
        powers[4]<-sum(pp<0.01)
        powers[5]<-sum(pp<0.005)
        powers<-powers/trials
        return(powers) }

simpowers(10000, 6, 160, -1, 2, 10)
```

For Using Simulation to Estimate the Power of a Warranty Example (in my Section 18):

```
install.packages("nlme") # you only have to install a package once
library(nlme)

simlogpay <- function(numclaims) {
        logpay <- rnorm(numclaims, mean=1, sd=1)
        logpay <- c(logpay, rnorm(numclaims, mean=2, sd=2) )
        logpay <- c(logpay, rnorm(numclaims, mean=3, sd=3) )
        logpay <- c(logpay, rnorm(numclaims, mean=4, sd=4) )
        return(logpay) }
simlogpays <- function(nstores, nclaims, sigmastore){
        lp <- rep(NA, times = 4*nstores*nclaims)
        for (i in 1: nstores){
        lp[4*nclaims*(i-1)+ 1:(4*nclaims)]<-simlogpay(nclaims) + rnorm(1, 0, sigmastore) }
        return(lp) }

createstore<-function(nstores, nclaims){
        st<-rep(1,4*nclaims)
        for (i in 2:nstores){
        st<-c(st,rep(i,4*nclaims))}
        return(st)}

createlag<-function(nstores, nclaims){
        lag<-c(rep(1,nclaims), rep(2,nclaims), rep(3,nclaims), rep(4,nclaims))
        for (i in 2:nstores){
        lag<-c(lag, rep(1,nclaims), rep(2,nclaims), rep(3,nclaims), rep(4,nclaims))}
        return(lag)}

testhet<-function(nstores, nclaims, sigmastore, lags, stores){
        logpay<-simlogpays(nstores, nclaims, sigmastore)
        m1<-lme(logpay~lags, random = ~1 | stores, method = "REML")
        # the 2nd model assumes that the variance of residuals is a power function of the lags
        wf<-varPower(form=~lags)
        m2<-lme(logpay~lags, random = ~1 | stores, method = "REML", weights = wf)
        pvalue<-anova(m1,m2)[2,9]
        return(pvalue)}

testhets <- function(trials, nstores, nclaims, sigmastore, lags, stores){
        pvalues <- rep(NA, times = trials)
        for (i in 1:trials){
        pvalues[i]<-testhet(nstores, nclaims, sigmastore, lags, stores)}
        return(pvalues)}
```

```
simpowers <- function(trials, nstores, nclaims, sigmastore){
        lags <-createlag(nstores, nclaims)
        stores<-createstore(nstores, nclaims)
        powers<- rep(NA, times = 5)
        pp<-testhets(trials, nstores, nclaims, sigmastore, lags, stores)
        powers[1]<-sum(pp<0.1)
        powers[2]<-sum(pp<0.05)
        powers[3]<-sum(pp<0.025)
        powers[4]<-sum(pp<0.01)
        powers[5]<-sum(pp<0.005)
        powers<-powers/trials
        return(powers) }

simpowers(10000, 3, 2, 1)
```