Applied Statistical Methods: Regression Analysis

Shizhe Chen

2020-04-17

Contents

Preface **5**

1 Linear regression with R **7**
1.1 Advertising data 7
1.2 Flu shot 8
1.3 Project STAR 10
1.4 Note 11

2 Estimation **13**

3 Sampling distribution **19**
3.1 Understanding sampling distribution via simulation 19
3.2 Shapes of sampling distributions 23
3.3 Small sample size 26

4 Statistical inference: Confidence Intervals **29**
4.1 Confidence interval 29
4.2 Prediction interval 32
4.3 Simultaneous confidence intervals/bands/regions 32

5 Statistical inference: Hypothesis testing **35**
5.1 Hypothesit testing 35
5.2 Multiple testing 38

6 Model diagnostics **41**
6.1 Residual plot 41
6.2 Remedies for non-linearity 42
6.3 Independence 42
6.4 Normality 43
6.5 Homoscedasticity 44
6.6 Influential Observations and Outliers 44

7 Multiple covariates **47**
7.1 Examples 47
7.2 Classification of variables 48

7.3 Least squares estimation 50
7.4 Underfitting and overfitting 52
7.5 Sampling distribution and inference 54

8 Model selection **57**
8.1 Criteria 57
8.2 Selection procedure 58

A R basics **59**
A.1 Basic objects in R 59
A.2 Summary statistics 60
A.3 Data structures 61
A.4 List 62
A.5 Functions in R 62
A.6 Miscellaneous 63

B Linear algebra **65**
B.1 Vector 65
B.2 Matrix 66
B.3 Other operations on vectors and matrices 71

C Simulation and visualization **73**
C.1 Simulation and visualization: univariate 73
C.2 Simulation and visualization: multivariate 75

Preface

This Gitbook contains code for STA 108. Lecture notes can be found in folder 'notes' or on the course website on Canvas. The Githbook is a work-in-progress, and materials in this Gitbook are updated constantly.

Chapter 1

Linear regression with R

Reading materials: Slides 3 - 11 in STA108_LinearRegression_S20.pdf.

Fitting a linear model is simple in R. The bare minimum requires you to know only two functions `lm()` and `summary()`. We will apply linear regression on three data set `advertising`, `flu shot`, and `Project STAR`.

1.1 Advertising data

This data is taken from An Introduction to Statistical Learning, by James et al. A brief description from Section 2.1 in ISL is provided below. You may read more about the data set in the free e-book.

> The Advertising data set consists of the sales of that product in 200 different markets, along with advertising budgets for the product in each of those markets for three different media: TV, radio, and newspaper… It is not possible for our client to directly increase sales of the product. On the other hand, they can control the advertising expenditure in each of the three media. Therefore, if we determine that there is an association between advertising and sales, then we can instruct our client to adjust advertising budgets, thereby indirectly increasing sales. In other words, our goal is to develop an accurate model that can be used to predict sales on the basis of the three media budgets.

It is important to note that we will avoid making any causal statements (i.e., increasing TV advertising budget will increase the sales by …) in our analysis. Causal inference on observational data is another major field in statistics. We will limit our discussion on association.

```
dat.advertising=read.csv('./data/advertising.csv');
# Use the code in Appendix C to visualize this data set
library(ggplot2)
library(GGally)
ggpairs(dat.advertising)
```

```
# What is X in this data frame?
dat.advertising=dat.advertising[,-1]
```

We will consider a simple linear regression model in this chapter. The only covariate/predictor/independent variable that we use here is the TV advertising budget. Therefore, the model is

$$y_i = x_i\beta_1 + \beta_0 + \epsilon_i, i = 1, \ldots, 200,$$

where y_i is the `sales` (1 unit = 1000 dollars) for the ith entry, x_i is the TV advertising budget for the ith entry, β_0 is the intercept term, and β_1 is the regression slope. In addition, we assume that the errors $\{\epsilon_i\}_{i=1}^{200}$ satisfy that $\epsilon_1, \ldots, \epsilon_{200}$ are independently and identically distributed (i.i.d.), $\mathbb{E}[\epsilon_i] = 0$ for $i = 1, 2, \ldots, 200$ and $\text{var}(\epsilon_i) = \sigma^2$ for $i = 1, 2, \ldots, 200$. Recall that we consider fixed design (i.e., x_i is not random) in this course for simplicity.

```
# Fit a simple linear regression
fit.advertising = lm(sales~TV+1,data=dat.advertising);
# Summarize the fitted results
summary(fit.advertising)
```

We see that $\hat{\beta}_1$ equals 0.05 and $\hat{\beta}_0$ equals 7.03. We can draw the line in the scatter plot.

How would you interpret the fitted slope and intercept?

```
# Draw the fitted line using ggplot2
ggplot(data = dat.advertising) +
  geom_point(mapping = aes(x = TV, y = sales)) +
  geom_line(data = fortify(fit.advertising ), aes(x = TV, y = .fitted),color='red')
```

```
# In fact, you can use the following code without fitting the lm()

ggplot(data = dat.advertising aes(x = TV, y = sales)) +
  geom_point() +
  geom_smooth(method = "lm")
```

```
# The band is a confidence band
```

1.2 Flu shot

The data come from the following paper: McDonald, C., Hiu, S., AND Tierney, W. (1992). Effects of computer reminders for influenza vaccination on morbidity during influenza epidemics. MD Computing 9, 304-312. (link).

Briefly, in this study, physicians were randomly selected to receive a letter encouraging them to inoculate patients at risk for flu. The treatment of interest is the actual flu shot, and the outcome is an indicator for flu-related hospital visits from the patients. In this data set, you can also find four important background covariates including gender, age, a chronic obstructive pulmonary disease (COPD) indicator, and a heart disease indicator. For further

details, read the original publication or Hirano, Imbens, Rubin and Zhou (2000) (link), or Chapter 25 in Imbens and Rubin (2015) (link).

```
dat.flu = read.table("./data/flu240.txt", header = TRUE)

# Take a look at size of the data before proceed:
dim(dat.flu)
colnames(dat.flu)
# Basic visulization:
ggpairs(dat.flu) # This might take a while..
```

```
# Do you think the plots are informative?

# One strategy: jittering

# Without jittering:
ggplot(dat.flu, aes(x=treatment.assigned, y=outcome)) + geom_point()
```

```
# With jittering:
ggplot(dat.flu, aes(x=treatment.assigned, y=outcome)) + geom_jitter(width = 0.25)
```

Again, we will consider a simple linear regression model for now. The only covariate/predictor/independent variable that we use here is the treatment assignment. Therefore, the model is

$$y_i = x_i\beta_1 + \beta_0 + \epsilon_i, i = 1, \ldots, 2891,$$

where y_i is the `outcome` (0 or 1) for the ith patient, x_i is the treatment assignment (0 or 1) for the ith patient's physician, β_0 is the intercept term, and β_1 is the regression slope. In addition, we assume that the errors $\{\epsilon_i\}_{i=1}^{2891}$ satisfy that $\epsilon_1, \ldots, \epsilon_{2891}$ are independently and identically distributed (i.i.d.), $\mathbb{E}[\epsilon_i] = 0$ for $i = 1, 2, \ldots, 2891$ and $\text{var}(\epsilon_i) = \sigma^2$ for $i = 1, 2, \ldots, 2891$. The following code is almost identical to those in the previous section.

How would you interpret the fitted slope and intercept?

Are there any violations to the assumptions on $\{\epsilon_i\}_{i=1}^{2891}$?

```
# Fit a simple linear regression
fit.flu= lm(outcome~treatment.assigned+1,data=dat.flu);
# Summarize the fitted results
summary(fit.flu)

# Draw the fitted line using ggplot2
ggplot(dat.flu, aes(x=treatment.assigned, y=outcome)) +
  geom_jitter(width = 0.25)+
  geom_smooth(method = "lm")
```

1.3 Project STAR

Tennesses Student/Teacher Achievement Ratio study (Project STAR) was conducted in the late 1980s to evaluate the effect of class size on test scores. This dataset has been used as a classic examples in many textbooks and research papers. Brieftly, the study randomly assigned students to small classes, regular classes, and regular classes with a teacher's aide. In order to randomize properly, schools were enrolled only if they had enough studybody to have at least one class of each type. Once the schools were enrolled, students were randomly assigned to the three types of classes, and one teacher was randomly assigned to one class. You can read more about the study on the Harvard dataverse (link) or Chapter 9 in Imbens and Rubin (2015) (link).

```
library(AER)
data("STAR")
dat.STAR=STAR; # Just to be consistent
# Take a look at size of the data before proceed:
dim(dat.STAR)
colnames(dat.STAR)
# With 47 variables, you might want to read the help file to see what these variables


# Basic visulization:
# ggpairs(dat.flu) # This might take a really long time ...

# We will consider the math scores and the class assignments in the second grade

# With jittering:
ggplot(dat.STAR, aes(x=star2, y=math2)) +
  geom_jitter(width = 0.25)
```

Again, we will consider a simple linear regression model for now. We will study the association between math scores and the class assignments in the second grade. Therefore, the model is

$$y_i = \beta_0 + x_{i,1}\beta_1 + x_{i,2}\beta_2 + \epsilon_i, i = 1, \ldots, 6065,$$

where y_i is the math score for the ith student, $x_{i,1}$ is an indicator (0 or 1) whether the ith student is in the small class, and $x_{i,1}$ is an indicator (0 or 1) whether the ith student is in the regular class with aide, β_0 is the intercept term, and β_1 and β_2 are the regression coefficients for $x_{i,1}$ and $x_{i,2}$, respectively. In addition, we assume that the errors $\{\epsilon_i\}_{i=1}^{6065}$ satisfy that $\epsilon_1, \ldots, \epsilon_{6065}$ are independently and identically distributed (i.i.d.), $\mathbb{E}[\epsilon_i] = 0$ for $i = 1, 2, \ldots, 6065$ and $\text{var}(\epsilon_i) = \sigma^2$ for $i = 1, 2, \ldots, 6065$. The sample size (6065) differs from the dimension of the original data set. The reason is that there are missing values in the data set. The available data will change we look at data from other grades.

The following code is almost identical to those in the previous section.

```r
# Fit a simple linear regression
fit.STAR= lm(math2~as.factor(star2)+1,data=dat.STAR);
# Summarize the fitted results
summary(fit.STAR)

# We can no longer draw a fitted line here...
```

1.4 Note

In the homework assignments and the midterm project, you will be asked to reproduce the analysis on the real and sythenetic data using your own functions. The hope is that, by reinventing the wheels, you will have a thorough understanding of linear regression.

Chapter 2

Estimation

Reading materials: Slides 12 - 22 in STA108_LinearRegression_S20.pdf

```
# Consider the advertising dataset
dat.advertising=read.csv('./data/advertising.csv');
plot(x=dat.advertising$TV, y=dat.advertising$sales, pch=16,cex=1.3,col='blue',xlab='x',y


# Suppose that we want to draw a line in this scatter plot
beta=c(4,0.03); # beta0=4; beta1=0.03; # Just two arbitrary numbers

# Visualize the line in the scatter plot
# This time we use the plotting functions in base R (i.e., not ggplot2)
# With the above function, we can reproduce the plot in Slide 13
plot(x=dat.advertising$TV, y=dat.advertising$sales, pch=16,cex=1.3,col='blue',xlab='x',y
abline(a=beta[1],b=beta[2],col='red',lwd=3)
```

```
# To reproduce the plot in Slide 13
# We need to find the corresponding point on the line for each data point
plot(x=dat.advertising$TV, y=dat.advertising$sales, pch=16,cex=1.3,col='black',xlab='x',
abline(a=beta[1],b=beta[2],col='red',lwd=3)
yout=dat.advertising$TV*beta[2]+beta[1];
for(i in 1:dim(dat.advertising)[1]){
  segments(x0=dat.advertising$TV[i], y0=yout[i],  y1 = dat.advertising$sales[i],col='blu
}
```

```
# What if we want to feed in different numbers?

# Write a function that can calculate x*beta1 + beta0

linear.model<-function(beta,covariate){
  # beta: a 2-vector, where the first entry is the intercept
  yout=covariate*beta[2]+beta[1]
```

```
  return(yout);
  # Note: this function only works with simple linear regression model
  # How would you generalize it?
}

fits=linear.model(beta=c(4,0.03),covariate=dat.advertising$TV);

plot(x=dat.advertising$TV, y=dat.advertising$sales, pch=16,cex=1.3,col='blue',xlab='x',y

points(fits~dat.advertising$TV,col='red')
```

```
# Furthermore, we can wrap up the plotting code into one function as well

plot.scatter.line<-function(beta,covariate,response){

  plot(x=covariate, y=response, pch=16,cex=1.3,col='blue',xlab='x',ylab='y')
  abline(a=beta[1],b=beta[2],col='red',lwd=3)

  yout=linear.model(beta=beta,covariate=covariate);
  for(i in 1:dim(dat.advertising)[1]){
    segments(x0=covariate[i], y0=yout[i],  y1 = response[i],col='blue')
  }
}



# We can now draw lines
plot.scatter.line(beta=c(4,0.03),covariate=dat.advertising$TV,response=dat.advertising$s
```

```
# We can see that the line is less than ideal

# Measuring the loss:
## We will move on to find the minimizer of the squared error loss

sum.of.squares<-function(beta,covariate,outcome){
  yout=linear.model(beta=beta,covariate=covariate);
  res=outcome-yout;
  sos= sum(res^2);
  return(sos)
}

# There are many ways to minimize the sum of squares

# 1, Use generic optimizer in R
fit.optim=optim(c(0,0),sum.of.squares,covariate=dat.advertising$TV,outcome=dat.advertisi
```

```
beta.hat.optim=fit.optim$par;

# 2. Use analytic form of the optimizer

fit.linear.model<-function(covariate,outcome){
  # I will write down the function for (multiple) linear regression here
  X=cbind(1,covariate);
  beta.fit=solve( t(X)%*%X )%*%t(X)%*%outcome;
  return(beta.fit)
}

beta.hat=fit.linear.model(covariate=dat.advertising$TV,outcome=dat.advertising$sales)

# 3. Write your own version of Newton-Raphson method, or gradient descent
# Not required or taught in this class

# We might want to check if our solution matches that from lm()

fit.advertising=lm(sales~TV+1,data=dat.advertising);

fit.advertising$coef
beta.hat
beta.hat.optim
```

```
# Summary of lm()
summary(fit.advertising)
```

```
# Decomposition of sum of squares
residual.sum.of.squares=sum.of.squares(beta=beta.hat,covariate=dat.advertising$TV,outcor
explained.sum.of.squares=sum((linear.model(beta=beta.hat,covariate=dat.advertising$TV)-m
total.sum.of.squares=sum((dat.advertising$sales-mean(dat.advertising$sales))^2)

# Check if the equality holds..
total.sum.of.squares
explained.sum.of.squares+residual.sum.of.squares

# But if we use identical()...
identical(total.sum.of.squares,explained.sum.of.squares+residual.sum.of.squares)


# Calculate the coefficient of determination
R.sq= explained.sum.of.squares/total.sum.of.squares;

# Calculate the Pearson sample correlation
```

```
cor.hat=cor(dat.advertising$TV,dat.advertising$sales)

# Verify the identity:
R.sq
cor.hat^2
```

```
# Q: what if these equalities only hold on this one data set? What would you do to rea

# One option is to use simulation to verify if the equality holds in every instance in
# Simulate N.sim instances
N.sim=10000;
beta=c(2,0.4);n=50;x=runif(n,min=-2,max=7.5);sigma=2;
out=matrix(0,nrow=N.sim,ncol=2)
for(i in 1:N.sim){
  Ey=x*beta[2]+beta[1];
  error.terms=sigma*rnorm(n);
  y=Ey+error.terms;
  beta.hat=fit.linear.model(covariate=x,outcome=y)

  # Copying code from the previous trunk:
  residual.sum.of.squares=sum.of.squares(beta=beta.hat,covariate=x,outcome=y)
  explained.sum.of.squares=sum((linear.model(beta=beta.hat,covariate=x)-mean(y))^2)
  total.sum.of.squares=sum((y-mean(y))^2)
  R.sq= explained.sum.of.squares/total.sum.of.squares;
  cor.hat=cor(x,y)
  out[i,]=c(R.sq,cor.hat)
}

plot(y=out[,1],x=out[,2],xlab='Correlation',ylab='Coef. of determination')
```

```
# It's better to wrap it up into one function:

N.sim=10000;
beta=c(2,0.4);n=50;x=runif(n,min=-2,max=7.5);sigma=2;

simulate.one.instance<-function(x,beta,sigma){
  n=length(x)
 Ey=x*beta[2]+beta[1];
  error.terms=sigma*rnorm(n);
  y=Ey+error.terms;
  beta.hat=fit.linear.model(covariate=x,outcome=y)

  # Copying code from the previous trunk:
  residual.sum.of.squares=sum.of.squares(beta=beta.hat,covariate=x,outcome=y)
  explained.sum.of.squares=sum((linear.model(beta=beta.hat,covariate=x)-mean(y))^2)
```

```
  total.sum.of.squares=sum((y-mean(y))^2)
  R.sq= explained.sum.of.squares/total.sum.of.squares;
  cor.hat=cor(x,y)
  out=c(R.sq,cor.hat)
  return(out)
}

set.seed(1)
sim=replicate(N.sim,simulate.one.instance(x,beta,sigma));

plot(y=sim[1,],x=sim[2,],xlab='Correlation',ylab='Coef. of determination')
```

```
# However, stimulation only tells us that the equality holds in the scenario that we d
# To establish the equality for all possible scenarios, we still have to use rigorous
# You can derive the property yourself if you are good at algebra
# Or search online to read others' proof and explanation
```

```
log.likelihood<-function(beta.sigma,covariate,outcome){
  yout=linear.model(beta=beta.sigma[-3],covariate=covariate);
  res=outcome-yout;
  lglklh=-sum( log(dnorm(x=res,mean=0,sd=beta.sigma[3])   ))
  return(lglklh)
}


# 1, Use generic optimizer in R
fit.lglklh.optim=optim(c(0,0,1),log.likelihood,covariate=dat.advertising$TV,outcome=dat.
fit.lglklh.optim
fit.optim
```

Chapter 3

Sampling distribution

3.1 Understanding sampling distribution via simulation

Reading materials: Slides 23 - 32 in STA108_LinearRegression_S20.pdf.

In this section, we assume a true data generating model in order to draw samples from the true popluation distribution. In particular, we assume that

$$y_i = x_i\beta_1 + \beta_0 + \epsilon_i,\ i = 1, \ldots, 50$$

where $\epsilon_i \sim \mathcal{U}(-2.5, 2, 5)$. We set $\beta_1 = 0.15$ and $\beta_0 = 20$.

```
### We will simulate synthetic data to understand the concept of sampling distribution

### To match our setup, we will generate the covarates and keep them fixed in the remain
set.seed(1)
n=50;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.true=c(20,0.15)
Ey= x*beta.true[2]+beta.true[1];

error.terms= (runif(n)-0.5)*5;
y=Ey+error.terms;
beta.hat=fit.linear.model(covariate=x,outcome=y);


plot(x=x,y=y,pch=16,cex=2)
abline(b=beta.hat[2],a=beta.hat[1],col='red',lwd=2)


### We can run the simulation by putting the above lines into a large for-loop
```

```
### We will wrap them up into one function

simulate.one.instance<-function(x,beta.true){
  n=length(x);
  Ey= x*beta.true[2]+beta.true[1];
  error.terms= (runif(n)-0.5)*5;
  y=Ey+error.terms;
  beta.hat=fit.linear.model(covariate=x,outcome=y);
  return(beta.hat)
}

### Set the number of replicates to be 10000
N.sim=1e4;
set.seed(1)
beta.sim=replicate(N.sim,simulate.one.instance(x,beta.true));


# Draw the histogram for the estimated intercept
hist(beta.sim[1,1,],xlab='Fitted intercept',main='')
abline(v=beta.true[1],lwd=3,col='red')
```

Recall from the lecture notes that the variance of $\hat{\beta}_1$ is

$$\text{var}\left(\hat{\beta}_1\right) = \frac{1}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sigma^2.$$

```
## Using the simulation results, we can verify the statement on Slide 26

### Expectation
mean(beta.sim[1,1,])-beta.true[1] # intercept
mean(beta.sim[2,1,])-beta.true[2] # slope

### Variance
sigma.sq = 5^2/12; # variance of the error: (runif(n)-0.5)*5;

# slope:
sigma.sq/sum(  (x-mean(x))^2  ) # theoretical value
var(beta.sim[2,1,]) # variance from the simulation


# intercept:
sigma.sq*sum(x^2)/sum(  (x-mean(x))^2  )/n # theoretical value
var(beta.sim[1,1,]) # variance from the simulation
```

Q: How would you verify the Gauss-Markove theorem in this simulation?

```
# A very simple simulation to compare the LSE with another linear unbiased estimator:
simulate.one.instance<-function(x,beta.true){
  n=length(x);
  Ey= x*beta.true[2]+beta.true[1];
  error.terms= (runif(n)-0.5)*5;
  y=Ey+error.terms;
  beta.hat=fit.linear.model(covariate=x,outcome=y);
  slope.hat=beta.hat[2];
  slope.dot= (y[n]-y[1])/(x[n]-x[1])
  return(c(slope.hat,slope.dot))
}

### Set the number of replicates to be 10000
N.sim=1e4;
set.seed(1)
slope.sim=replicate(N.sim,simulate.one.instance(x,beta.true));

apply(slope.sim,1,mean) # both are unbiased
apply(slope.sim,1,var) # variance of LSE is smaller
```

```
## Estimate the variance of errors using the residuals
set.seed(1)
error.terms= (runif(n)-0.5)*5;
y=Ey+error.terms;
beta.hat=fit.linear.model(covariate=x,outcome=y);

residual.sum.of.squares=sum.of.squares(beta=beta.hat,covariate=x,outcome=y)

sigma.sq.hat=residual.sum.of.squares/(n-2) # estimates from one instance

## Run a simulation to verify the claim

## NOTE: we will use the same function name
simulate.one.instance<-function(x,beta.true){
  n=length(x);
  Ey= x*beta.true[2]+beta.true[1];
  error.terms= (runif(n)-0.5)*5;
  y=Ey+error.terms;
  beta.hat=fit.linear.model(covariate=x,outcome=y);
  residual.sum.of.squares=sum.of.squares(beta=beta.hat,covariate=x,outcome=y)

sigma.sq.hat=residual.sum.of.squares/(n-2) # estimates from one instance

  return(sigma.sq.hat)
```

```
}

### Set the number of replicates to be 10000
N.sim=1e4;
set.seed(1)
sigma.sq.hat.sim=replicate(N.sim,simulate.one.instance(x,beta.true));

mean(sigma.sq.hat.sim)
sigma.sq
```

```
## To wrap this up
## For any new fits, we can estiamte the variance and standad errors of the estimators
set.seed(1)
error.terms= (runif(n)-0.5)*5;
y=Ey+error.terms;
beta.hat=fit.linear.model(covariate=x,outcome=y);

estimate.sigma.sq<-function(beta,covariate,outcome){
   residual.sum.of.squares=sum.of.squares(beta=beta,covariate=covariate,outcome=outcome)
   n=length(outcome)
   sigma.sq.hat=residual.sum.of.squares/(n-2)
   return(sigma.sq.hat)
}

estimate.coef.var<-function(beta,covariate,outcome){
  sigma.sq.hat=estimate.sigma.sq(beta,covariate,outcome)
  var.hat.beta=beta;
  var.hat.beta[2]=sigma.sq.hat/sum(  (covariate-mean(covariate))^2  )
  n=length(outcome)
  var.hat.beta[1]=sigma.sq.hat*sum(covariate^2)/sum((covariate-mean(covariate))^2  )/n
  return( var.hat.beta)
}

estimate.coef.sd<-function(beta,covariate,outcome){
  var.hat.beta=estimate.coef.var(beta,covariate,outcome)
 sd.hat.beta=sqrt(var.hat.beta);
  return(sd.hat.beta)
}
estimate.coef.sd(beta=beta.hat,covariate=x,outcome=y)

## Compare your numbers with the output from lm()
summary(lm(y~x+1))
```

3.2 Shapes of sampling distributions

Reading materials: Slide 33 in STA108_LinearRegression_S20.pdf.

```
# There can be multiple distributions with the same mean and variance. Consider three

xgrid = seq(from = -5, to = 5, by=0.01);
norm_density=dnorm(xgrid,mean=0,sd=sqrt(2));
unif_density=dunif(xgrid,min=-sqrt(6),max=sqrt(6));

xbin=c(-sqrt(2),sqrt(2));
ybin=c(0.5,0.5);


plot(norm_density~xgrid,type="l",col='red',lwd=3,ylim=c(0,0.52),xlab="z",ylab="density")
lines(unif_density~xgrid,col='black',lwd=3)

segments(xbin[1],0,xbin[1],ybin[1],col='green',lwd=10)
segments(xbin[2],0,xbin[2],ybin[2],col='green',lwd=10)
legend(x=2.5,y=0.5,legend=c("Uniform","Normal","Binary"),lwd=2,col=c('black','red','gree
```

3.2.1 Asymptotic distribution

Reading materials: Slides 34 - 48 in STA108_LinearRegression_S20.pdf.

The L-F central limit theorem claims that the asymptotic distribution for $\frac{\hat{\beta}_1-\beta_1}{\hat{\sigma}/S_{xx}^{1/2}}$ is $\mathcal{N}(0,1)$.

```
## Edit the simulation code
### This time with large n (n=500)
### Three different versions of errors
### Unequal variances
simulate.one.instance<-function(x,beta.true,error.type){
  n=length(x);
  Ey= x*beta.true[2]+beta.true[1];
  vars=0.1+runif(n)*2; # unequal variance
  if(error.type=='uniform'){
    error.terms= (runif(n)-0.5)*sqrt(vars);

  }else if(error.type=='Bernoulli'){
    error.terms=(rbinom(n,size=1,prob=0.5)-0.5)*sqrt(vars);
  }else{
    error.terms= rnorm(n)*sqrt(vars);
  }
  y=Ey+error.terms;
  beta.hat=fit.linear.model(covariate=x,outcome=y);
```

```
  residual.sum.of.squares=sum.of.squares(beta=beta.hat,covariate=x,outcome=y)

sigma.sq.hat=residual.sum.of.squares/(n-2) # estimates from one instance

  beta.hat.std =(beta.hat[2]-beta.true[2])/sqrt(sigma.sq.hat)*sqrt( sum( (x-mean(x))^2 )

  return(beta.hat.std)
}

### Set the number of replicates to be 10000
N.sim=1e4;
set.seed(1)

n=500;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.hat.std.uniform=replicate(N.sim,simulate.one.instance(x,beta.true,error.type='unif
beta.hat.std.Bernoulli=replicate(N.sim,simulate.one.instance(x,beta.true,error.type='Be
beta.hat.std.normal=replicate(N.sim,simulate.one.instance(x,beta.true,error.type='norma
```

```
# Draw the density plot

density.uniform=density( beta.hat.std.uniform);
density.Bernoulli=density( beta.hat.std.Bernoulli);
density.normal=density( beta.hat.std.normal);

# Theoretical density:
xgrid = seq(from=-4,to=4,length.out=100);
normal.pdf= dnorm(xgrid)

plot(density.uniform,xlab="Standardized intercept",col='red',lwd=3,ylim=c(0,0.5),main=''
lines(density.Bernoulli,col='green',lwd=3,lty=2)
lines(density.normal,col='blue',lwd=3,lty=2)
lines(normal.pdf~xgrid,lwd=2,lty=3)
legend(x=2,y=0.46,legend=c("Uniform errors","Bernoulli errors","Normal errors", "N(0,1)'
```

```
## The L-F CLT allows us to write down the asymptotic distribution in closed-form
## The bootstrap method is an alternative when analytic solution is not available
## Note: Bootstrap is not that useful for linear regression, but has proven to be a powe

set.seed(1)
# Generate one set of data:
n=500;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
Ey= x*beta.true[2]+beta.true[1];
vars=0.1+runif(n)*2; # unequal variance
```

```
error.terms= (runif(n)-0.5)*sqrt(vars);
y=Ey+error.terms;
beta.hat=fit.linear.model(covariate=x,outcome=y);


# Now write a function for bootstrap
boot.one.instance<-function(covariate,outcome){
  n=length(outcome);
  sample_indices = sample(1:n,n,replace=TRUE) # sampling with replacement
  covariate.boot= covariate[sample_indices]; outcome.boot= outcome[sample_indices];

  beta.hat=fit.linear.model(covariate=covariate.boot,outcome=outcome.boot);
  return(beta.hat[2])
}

set.seed(1)

B=1e5;
beta.hat.boot=replicate(B,boot.one.instance(covariate=x,outcome=y));
```

```
## Compare the bootstrap distribution of the standardized slope against that from the CL


density.boot=density((beta.hat.boot-mean(beta.hat.boot))/sd(beta.hat.boot));
# Theoretical density (from CLT)
xgrid = seq(from=-4,to=4,length.out=100);
normal.pdf= dnorm(xgrid)

plot(density.boot,xlab="Standardized intercept",col='red',lwd=3,ylim=c(0,0.5),main='',lt
lines(normal.pdf~xgrid,lwd=2,lty=3)
legend(x=2,y=0.46,legend=c("Bootstrap", "N(0,1)"),lwd=3,col=c('red','green','blue','blac
```

```
## Compare the distributions using the raw bootstrap distribution, and the distribution


density.boot=density(beta.hat.boot);
# Theoretical density (from CLT)
xgrid = seq(from=min(beta.hat.boot),to=max(beta.hat.boot),length.out=100);
normal.pdf= dnorm(xgrid,mean=mean(beta.hat.boot),sd=sd(beta.hat.boot))

plot(density.boot,xlab="Standardized intercept",col='red',lwd=3,main='',lty=2)
lines(normal.pdf~xgrid,lwd=2,lty=3)
legend(x=0.15,y=50,legend=c("Empirical", "Moment-based"),lwd=3,col=c('red','green','blue
```

3.3 Small sample size

Reading materials: Slides 49 - 57 in STA108_LinearRegression_S20.pdf.

When the sample size is small, the asymptotic distribution is not close to the true sampling distribution. Neither the central limit theorem or the bootstrap distribution are good approximation of the asymptotic distribution.

```
# Simulation with small sample size
# We will demonstrate the case with 5 samples
# You will find the code in this trunk almost identical with previous code

# Generate one set of data:
set.seed(1)
n=10;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.true=c(20,0.15)
Ey= x*beta.true[2]+beta.true[1];

error.terms= rnorm(n)*5; # Normal errors!
y=Ey+error.terms;
beta.hat=fit.linear.model(covariate=x,outcome=y);
plot(x=x,y=y,pch=16,cex=2)
abline(b=beta.hat[2],a=beta.hat[1],col='red',lwd=2)


## Simulate the true sampling distribution:
simulate.one.instance<-function(x,beta.true){
  n=length(x);
  Ey= x*beta.true[2]+beta.true[1];
  error.terms= (runif(n)-0.5)*5;
  y=Ey+error.terms;
  beta.hat=fit.linear.model(covariate=x,outcome=y);
  return(beta.hat)
}

### Set the number of replicates to be 10000
N.sim=1e4;
true.beta.hat=replicate(N.sim,simulate.one.instance(x,beta.true));

### Obtain the mean and standard deviation using CLT:
residual.sum.of.squares=sum.of.squares(beta=beta.hat,covariate=x,outcome=y)
sigma.sq.hat=residual.sum.of.squares/(n-2) # estimates from one instance


### Obtain the bootstrap distribution
```

```
B=1e5;
boot.beta.hat=replicate(B,boot.one.instance(covariate=x,outcome=y));
```

```
density.true=density( (true.beta.hat[2,1,]-mean(true.beta.hat[2,1,]))/sd(true.beta.hat[2

density.boot=density((boot.beta.hat-mean(boot.beta.hat))/sd(boot.beta.hat));
# Theoretical density (from CLT)
xgrid = seq(from=-4,to=4,length.out=100);
normal.pdf= dnorm(xgrid)


plot(density.true,xlab="Estimated intercept",lwd=3,ylim=c(0,0.5),xlim=c(-4,4),main='',lt
lines(density.boot,col='green',lwd=3,lty=2)
lines(normal.pdf~xgrid,col='red',,lwd=2,lty=2)
legend(x=2,y=0.5,legend=c("True","Bootstrap","CLT"),lwd=3,col=c('black','green','red'),l
```

```
## With very few samples, we have to exploit the parametric assumption
## In this case, we take advantage of the normality assumption on the errors


t.pdf= dt(xgrid,df=n-2)

plot(density.true,xlab="Estimated intercept",lwd=3,ylim=c(0,0.5),xlim=c(-4,4),main='',lt
lines(density.boot,col='green',lwd=3,lty=2)
lines(normal.pdf~xgrid,col='red',,lwd=2,lty=2)

lines(t.pdf~xgrid,col='blue',,lwd=2,lty=2)
legend(x=2,y=0.5,legend=c("True","Bootstrap","CLT",'Student t'),lwd=3,col=c('black','gre
```

```
## The Student t distribution grows more similar to the standard normal as its degrees o

t_dist=dt(xgrid,df=1);
plot(t_dist~xgrid,xlab="beta 1 hat",col=rgb(1,0,0,0.1),lwd=3,type="l",ylim=c(0,0.5),xlim

for(i in 2:10){
t_dist=dt(xgrid,df=i);

lines(t_dist~xgrid,col=rgb(1,0,0,i/10),lwd=3)

}

legend(x=2,y=0.46,legend=c("Student t", "N(0,1)"),lwd=3,col=c('red','green'))

lines(normal.pdf~xgrid,col=rgb(0,1,0,0.4),lwd=5)
```

Chapter 4

Statistical inference: Confidence Intervals

Reading materials: Slides 57 - 72 in STA108_LinearRegression_S20.pdf.

4.1 Confidence interval

```
## We will look at synthetic data here, because we have control over the truth
set.seed(1)
n=50;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.true=c(20,0.15)
Ey= x*beta.true[2]+beta.true[1];
error.terms= rnorm(n)*5;
y=Ey+error.terms;
```

```
# For a given confidence level alpha, construct a 100(1-alpha)% confidence interval

alpha=0.023;
# There is a function confint() in R
fit.lm=lm(y~x+1);
confint(fit.lm,level=1-alpha)
```

```
# We will implement our own version

## We actually have most of the part ready
beta.hat=fit.linear.model(covariate=x,outcome=y);
beta.sd=estimate.coef.sd(beta=beta.hat,covariate=x,outcome=y);

## The only missing piece is the quantile
conf.int.quantile<-function(alpha,type,...){
```

```
  if(type=="t"){
    out=qt(c(1-alpha/2,alpha/2), ... )
  }else if (type=="normal"){
    out=qnorm(c(1-alpha/2,alpha/2), ... )
  }
  return(out)
}
quants<-conf.int.quantile(alpha,type='t',df=n-2)

beta.hat%*%c(1,1)-beta.sd%*%quants

## Compare with the output from confint
confint(fit.lm,level=1-alpha)

## How about using bootstrap to construct the CIs?
boot.fit<-function(covariate,outcome){
  n=length(outcome);
  sample_indices = sample(1:n,n,replace=TRUE) # sampling with replacement
  covariate.boot= covariate[sample_indices]; outcome.boot= outcome[sample_indices];

  beta.hat=fit.linear.model(covariate=covariate.boot,outcome=outcome.boot);
  return(t(beta.hat ))
}
B=1e5;
beta.hat.boot=replicate(B,boot.fit(covariate=x,outcome=y));

dim(beta.hat.boot)
apply(beta.hat.boot[1,,],1,quantile,probs=c(alpha/2,1-alpha/2))

# We can wrap this up into a function
conf.int<-function(alpha,type,covariate,outcome,B=1e5){

  beta.hat=fit.linear.model(covariate,outcome);
  beta.sd=estimate.coef.sd(beta=beta.hat,covariate,outcome);
  if(type=='bootstrap'){
    beta.hat.boot=replicate(B,boot.fit(covariate,outcome));
    out=t(apply(beta.hat.boot[1,,],1,quantile,probs=c(alpha/2,1-alpha/2)));
  }else if(type=='t'){
    quants<-conf.int.quantile(alpha,type='t',df=n-2)
    out=beta.hat%*%c(1,1)-beta.sd%*%quants;
  }else{
    quants<-conf.int.quantile(alpha,type='normal')
    out=beta.hat%*%c(1,1)-beta.sd%*%quants;
  }
```

```
  colnames(out)=c( paste(round(alpha*50,digits=3),'%'), paste(100-round(alpha*50,digits
  return(out)
}
conf.int(alpha=alpha,type='t',covariate=x,outcome=y)
confint(fit.lm,level=1-alpha)
```

```
## Verify the coverage of the confidence invervals
simulate.one.instance<-function(x,beta.true,alpha){
  n=length(x);
  Ey= x*beta.true[2]+beta.true[1];
  error.terms= rnorm(n)*5;
  y=Ey+error.terms;
  CIs=conf.int(alpha=alpha,type='t',covariate=x,outcome=y);
  return(CIs)
}

N.sim=1e4;
set.seed(1)
n=50;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.true=c(20,0.15)
sim.CIs=replicate(N.sim,simulate.one.instance(x,beta.true,alpha));
```

```
## Visualize and verify the coverage for the slope
coverage=sum(sim.CIs[2,1,]<beta.true[2] & sim.CIs[2,2,]>beta.true[2])/N.sim;
1-alpha

plot(0,0,col="white",xlim=c(min(sim.CIs[2,1,]),max(sim.CIs[2,2,])),ylim=c(0,20),xlab="SI
abline(v=beta.true[2],lwd=3,col='red')
for(i in 1:19){
  segments(sim.CIs[2,1,i],i,sim.CIs[2,2,i],i,lwd=3)
}
```

4.1.1 Determine cutoffs

We can definite multiple confidence intervals of the same confidence levels for on the same sampling distribution.

```
alpha=0.05;

cutoffs=c(0.01,0.025,0.049);

colorlist=c('red','green','blue');
leg_text=c("1","2","3");
```

```
normal.pdf= dnorm(xgrid)
plot(normal.pdf~xgrid,xlab="beta 1 hat",ylab="Density",col=rgb(0,0,0,1),lwd=3,type="l",y

for(i in 1:length(cutoffs)){
lower_bound = qnorm(cutoffs[i]);
upper_bound = qnorm(1-alpha+cutoffs[i]);
segments(lower_bound,0,lower_bound,0.5,col=colorlist[i],lwd=3)
segments(upper_bound,0,upper_bound,0.5,col=colorlist[i],lwd=3)
segments(lower_bound,i*0.1,upper_bound,i*0.1,col=colorlist[i],lwd=3)
leg_text[i] =paste(cutoffs[i],"to", 1-alpha+cutoffs[i], "(length:", signif(upper_bound-l
}
legend(x=-5,y=0.5,legend=leg_text,lwd=3,col=colorlist)
```

4.2 Prediction interval

```
set.seed(1)
n=50;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.true=c(20,0.15)
Ey= x*beta.true[2]+beta.true[1];
error.terms= rnorm(n)*5;
y=Ey+error.terms;
fit.lm=lm(y~as.vector(x)+1);

alpha=0.05;
dat.new = data.frame(x=c(10,20));

# Prediction interval (assume normality)
predict(fit.lm,newdata=dat.new,interval="prediction",level=1-alpha)

# and confidence interval
predict(fit.lm,newdata=dat.new,interval="confidence",level=1-alpha)
```

```
# Implementation is similar to our conf.int function, and is thus skipped.
# You can verify the coverage by modifying the code above.
```

4.3 Simultaneous confidence intervals/bands/regions

```
## We will return to the advertising data
fit.lm=lm(sales~TV+1,data=dat.advertising)
fit.lm.sum=summary(fit.lm)
alpha=0.05;
```

```
xgrid_pred=seq(from=5,to=50,by=2.5);
dat.new = data.frame(TV=xgrid_pred);
pointwise=predict(fit.lm,newdata=dat.new,interval="confidence",level=1-alpha)

# Bonferroni corrected:
Bonf=predict(fit.lm,newdata=dat.new,interval="confidence",level=1-alpha/length(xgrid_pre


# Working-Hotelling
fstat= qf(1-alpha,2,length(dat.advertising$TV)-2);
xdense=seq(from=0,to=50,by=0.1);
Sxx= sum( (dat.advertising$TV-mean(dat.advertising$TV))^2);
pivot=sqrt(2*fstat)*fit.lm.sum$sigma*sqrt(1/length(dat.advertising$TV)+(xdense-mean(dat
ylb=fit.lm$coefficients[2]*xdense+fit.lm$coefficients[1]-pivot;
yub=fit.lm$coefficients[2]*xdense+fit.lm$coefficients[1]+pivot;


plot(dat.advertising$sales~dat.advertising$TV,pch=16,xlim=c(0,50),ylim=c(4,12),xlab='TV'
abline(b=fit.lm$coefficients[2],a=fit.lm$coefficients[1],lwd=3)
for(i in 1:length(xgrid_pred)){

segments(xgrid_pred[i]-1,Bonf[i,2],xgrid_pred[i]-1,Bonf[i,3],lwd=3,col='green')
segments(xgrid_pred[i]-1,pointwise[i,2],xgrid_pred[i]-1,pointwise[i,3],lwd=3,col='red')

}
lines(ylb~xdense,col='blue',lwd=3)

lines(yub~xdense,col='blue',lwd=3)
legend(x=32,y=6,legend=c("Pointwise", "Bonferroni", "W-H band"), lwd=3, col=c("red","gre
```

Chapter 5

Statistical inference: Hypothesis testing

Reading materials: Slides 73 - 91 in STA108_LinearRegression_S20.pdf.

We will continue working on synthetic data in this chapter.

```
set.seed(1)
n=50;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.true=c(20,0.15)
Ey= x*beta.true[2]+beta.true[1];
error.terms= rnorm(n)*2; # Normal errors, in order to verify the performance of the t-
# You can change it to other distributions
y=Ey+error.terms;
```

5.1 Hypothesit testing

```
# The lm() and summary() functions will automatically calculate the test statistics an
fit.lm<-lm(y~x+1);fit.lm.summary<-summary(fit.lm)
fit.lm.summary
```

```
# We will create our own function for calculating the "t value"
source('sources.r') # load the functions we have written

beta.hat=fit.linear.model(covariate=x,outcome=y)
beta.hat.sd=estimate.coef.sd(beta=beta.hat,covariate=x,outcome=y)

# The test statistics for H0: beta1=0, and H0: beta0=0 are
beta.hat.t = (beta.hat-0)/beta.hat.sd;
```

```
calculate.t<-function(covariate,outcome){
  beta.hat=fit.linear.model(covariate=covariate,outcome=outcome)
  beta.hat.sd=estimate.coef.sd(beta=beta.hat,covariate=covariate,outcome=outcome)
  beta.hat.t = (beta.hat-0)/beta.hat.sd;
  return(beta.hat.t)
}

# The p-values using the Student t distribution are
2*apply(cbind(pt(-abs(beta.hat.t),df=n-2),(1-pt(abs(beta.hat.t),df=n-2))),1,min)

# We can also calculate the p-values using CLT
2*apply(cbind(pnorm(-abs(beta.hat.t)),(1-pnorm(abs(beta.hat.t)))),1,min)

# Or to use the bootstrap
# This part is NOT required
# It is recommended to read more about bootstrap if you want to use it for hypothesis
# since the empirical bootstrap procedure is not the most efficient one for this purpo

beta.hat.boot=replicate(1e5,boot.fit(covariate=x,outcome=y));
pval=numeric(length(beta.hat));
for(i in 1:length(beta.hat)){
  boot.est=beta.hat.boot[1,i,]
  pval[i]=2*min( mean(0<boot.est), mean(0>boot.est) )
}

## Wrap up the above code into one function
calculate.pvalue<-function(covariate,outcome,type){
  beta.hat.t=calculate.t(covariate,outcome);
  if (type=='t'){
    n=length(outcome);
    pval=2*apply(cbind(pt(beta.hat.t,df=n-2),(1-pt(beta.hat.t,df=n-2))),1,min);

  }else if (type=='z'){
    pval=2*apply(cbind(pnorm(-abs(beta.hat.t)),(1-pnorm(abs(beta.hat.t)))),1,min);

  }else if (type == 'bootstrap'){

    beta.hat=fit.linear.model(covariate=covariate,outcome=outcome)
    beta.hat.boot=replicate(1e5,boot.fit(covariate=covariate,outcome=outcome));
    pval=numeric(length(beta.hat));
    for(i in 1:length(beta.hat)){
      boot.est=beta.hat.boot[1,i,]
      pval[i]=2*min( mean(0<boot.est), mean(0>boot.est) )
    }
```

```
  }
  return(pval)
}
```

```
## Write a simulation to verify the equivalence between type I error rate and significan
alpha=0.03; # significance level
simulate.one.instance<-function(x,beta.null,alpha,type){
  n=length(x);
  Ey= x*beta.null[2]+beta.null[1];
  error.terms= rnorm(n)*2;
  y=Ey+error.terms;
  pval=calculate.pvalue(covariate=x,outcome=y,type=type);
  rej.flag= pval[2]<alpha
  return(rej.flag)
}

N.sim=1e4;
set.seed(1)
n=50;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.null=c(beta.true[1],0)
sim.typeI=replicate(N.sim,simulate.one.instance(x,beta.null,alpha,type='t'));
mean(sim.typeI)
```

```
## Assess the power
N.sim=1e4;
set.seed(1)
n=50;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
powers<-numeric(3)
for(a in 1:3){
  beta.alt=c(beta.true[1],a/10)
  sim.rej=replicate(N.sim,simulate.one.instance(x,beta.alt,alpha,type='t'));
  powers[a]<-mean(sim.rej)

}
powers
```

```
## Permutation test for testing H0: beta1 = 0 (slope is zero)

permutation.test<-function(covariate,outcome){
  n=length(outcome);
  sample_indices = sample(1:n,n,replace=FALSE) # sampling without replacement
  covariate.perm= covariate[sample_indices]; outcome.perm= outcome[sample_indices];
```

```
  beta.hat.t=calculate.t(covariate.perm,outcome.perm)

  return(beta.hat.t[2])
}


set.seed(1)
n=50;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.true=c(20,0.15)
Ey= x*beta.true[2]+beta.true[1];
error.terms= rnorm(n)*2; # Normal errors, in order to verify the performance of the t-
# You can change it to other distributions
y=Ey+error.terms;

beta.hat.t.perm=replicate(1e5,permutation.test(covariate=x,outcome=y));
beta.hat.t=calculate.t(x,y)
2*min( mean(beta.hat.t[2]<beta.hat.t.perm),mean(beta.hat.t[2]>beta.hat.t.perm))
```

```
## F-test can be done using the anova() function in R
## It will be left as an exercise for you to write your version of F test
set.seed(1)
n=50;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
beta.true=c(20,0.15)
Ey= x*beta.true[2]+beta.true[1];
error.terms= rnorm(n)*2; # Normal errors, in order to verify the performance of the t-
# You can change it to other distributions
y=Ey+error.terms;

full_model=lm(y~x+1);
reduced_model=lm(y~1);
anova(reduced_model,full_model)
summary(lm(y~x+1))
```

5.2 Multiple testing

```
# To discuss multiple testing, we generate 1000 outcomes, among which 100 are truly as

set.seed(1)
n=50;m=1000;
x=as.matrix(rnorm(n,mean=10,sd=2),ncol=n);
Y=matrix(0,nrow=n,ncol=m)
```

```
beta.true=c(20,0.6)
for(i in 1:m){
Ey= x*beta.true[2]*(i<101)+beta.true[1];
error.terms= rnorm(n)*2;
Y[,i]=Ey+error.terms;
}

# Conduct the same test
pvalues=numeric(m)
for (i in 1:m){
  pval=calculate.pvalue(covariate=x,outcome=Y[,i],type='z');
  pvalues[i]<-pval[2];
}
```

```
## Bonferroni correction
alpha=0.05;
alpha.Bonf=alpha/m; # devided by the number of hypotheses
rej.flag=pvalues<alpha.Bonf;

## True positive
sum(rej.flag & ((1:m)<101))
## False positive
sum(rej.flag & ((1:m)>100))
## True negative
sum(!rej.flag & ((1:m)>100))

## False negative
sum(!rej.flag & ((1:m)<101))
```

```
## Benjamini-Hochberg procedure for FDR control

pvalues.sorted<-sort(pvalues,index.return=T)
alpha.BH<- (1:m)*alpha/m;
k.hat<-max(which(pvalues.sorted$x<alpha.BH))

rej.flag.BH=numeric(m);
rej.flag.BH[pvalues.sorted$ix[1:k.hat]]=1;



## True positive
sum(rej.flag.BH & ((1:m)<101))
## False positive
sum(rej.flag.BH & ((1:m)>100))
## True negative
```

```r
sum(!rej.flag.BH & ((1:m)>100))

## False negative
sum(!rej.flag.BH & ((1:m)<101))

## False positive rate

sum(rej.flag.BH & ((1:m)>100))/sum(rej.flag.BH)

# Run a simulation to verify this
```

Chapter 6

Model diagnostics

Reading materials: Slides 92 - 100 in STA108_LinearRegression_S20.pdf.

Note: we will use existing functions in R for model diagnostics in this chapter. However, in your midterm report, you are still required to implement your own tools for model diagnostics.

6.1 Residual plot

```
## We can extract residual from an `lm()` object.

fit.lm = lm(sales~TV+1,data=dat.advertising); # Fit the linear regression
resid= fit.lm$residuals;

plot(resid~dat.advertising$TV,pch=16,col='red',main='Residual plot')
abline(h=0,lwd=3)
```

Why don't we draw the residual plot as residuals v.s. the response? This is because the two quantities are surely positively correlated, and thus is hard to extract any useful information from the plot.

```
plot(resid~dat.advertising$sales,pch=16,col='red',main='Wrong plot')
abline(h=0,lwd=3)
```

How about drawing residuals against the fitted values? This is a good choice to detect certain anomalies.

```
par(mfrow=c(1,2))
plot(resid~dat.advertising$TV,pch=16,col='red',main='Residual plot')
abline(h=0,lwd=3)
plot(resid~fit.lm$fitted.values,pch=16,col='red',main='Residual plot',xlab='Fitted value
abline(h=0,lwd=3)
```

```
par(mfrow=c(1,1))
```

6.2 Remedies for non-linearity

From the residual plot, we see that there exist some non-linearity between the residuals and the covariate, especially when the TV is number is small. This could suggest that the relationship between TV and sales is nonlinear. Typical form of nonlinearity takes form as $\exp(x)$, $x^{1/2}$, $\log(x)$, x^2, etc. We may be able to guess the nonlinearity from the residual plot, or use model selection to pick the best nonlinear function, if there is not scientific knowledge on the relationship. In this example, we will fit two regression $y \sim \log(x)\beta_1 + \beta_0$ and $y \sim x^{1/2}\beta_1 + \beta_0$.

```
fit.log = lm(sales~log(TV)+1,data=dat.advertising); # Fit the linear regression
dat.advertising$sqrtTV=dat.advertising$TV^{1/2};
fit.sqrt = lm(sales~sqrtTV+1,data=dat.advertising); # Fit the linear regression

#par(mfrow=c(3,1))
plot(fit.log$residuals~log(dat.advertising$TV),pch=16,col='red',main='Residual plot (log
abline(h=0,lwd=3)

plot(fit.lm$residuals~dat.advertising$TV,pch=16,col='red',main='Residual plot (TV)',xlab
abline(h=0,lwd=3)

plot(fit.sqrt$residuals~dat.advertising$sqrtTV,pch=16,col='red',main='Residual plot (sq.
abline(h=0,lwd=3)
#par(mfrow=c(1,1))
```

6.3 Independence

In the `advertising` data set, the residual plot does not tell us whether certain data points are correlated or not.

```
plot(fit.lm$residuals~dat.advertising$TV,pch=16,col='red',main='Residual plot')
abline(h=0,lwd=3)
```

We will consider a model where we know the errors are correlated, and show that they may still be hard to recognized in practice. We consider a model with autoregressive error, AR(1), which is widely used in time-series data analysis (e.g., financial data). We consider a model where, for $i = 1, 2, \ldots, n$,

$$y_i = x_i\beta_1 + \beta_0 + \epsilon_i,$$

where $\epsilon_i = 3\epsilon_{i-1}/4 + z_i/4$ and $z_i \sim \mathcal{N}(0, 1)$. Here i represents some unit of time (e.g., months, years, days).

```
n=100;
set.seed(1);
z=rnorm(n);
epsilon= numeric(n);
for(i in 1:n){
if (i==1){
epsilon[i]=z[i];

}else{
epsilon[i]=3*epsilon[i-1]/4+z[i]/4;
}
}
x=rnorm(n)*10+5;
y=x*2+1+z;

ARfits = lm(y~x+1);


par(mfrow=c(1,2))
plot(ARfits$residuals~x,pch=16,col='red',main='Residual plot 1')
abline(h=0,lwd=3)

plot(ARfits$residuals~c(1:n),pch=16,col='red',main='Residual plot 2',xlab="Index i")
abline(h=0,lwd=3)

par(mfrow=c(1,1))
```

6.4 Normality

It is also hard to see normality based on the residual plot. We can use a Quantile-Quantile plot to check if the errors are normally distributed. In the Q-Q plot, we draw the quantiles of residuals against the quantiles of the thoeretical quantiles from a normal distribution. The $100(i/n)\%$th quantile of the residuals is defined as the ith smallest residual.

Non-normal distributions in Q-Q plots.

```
n=500;
distributions=matrix(0,n,4)
distributions[,1] =-exp(rnorm(n));
distributions[,2] =exp(rnorm(n));
distributions[,3]=rt(n,df=3);
distributions[,4]=runif(n);

titles = c('Left skewed', 'Right skewed', 'Heavy-tailed', 'Light-tailed' )
```

```
par(mfrow=c(2,2))
for ( i in 1:4){
this_dist=sort(distributions[,i]);
normal_mean =mean(this_dist);normal_sd = sd(this_dist);
this_dist=(this_dist - normal_mean)/normal_sd;
normal_quantiles = qnorm( (1:length(this_dist))/length(this_dist));

plot(this_dist~normal_quantiles,pch=16,col='red',main=titles[i],xlab='Normal quantiles',
abline(a=0,b=1,lwd=3)
}
```

6.5 Homoscedasticity

We can see clearly that the residuals have wider spans when TV is larger, which suggested increasing variance. We can stablize the variance by transforming the response variable $y^{1/2}$, $\log(y)$, etc. You can also use the Box-Cox transformation to find the most appropriate function.

```
fit.ylog=lm(log(sales)~TV+1,data=dat.advertising)
fit.ysqrt=lm(sqrt(sales)~TV+1,data=dat.advertising)
#par(mfrow=c(3,1))
plot(fit.lm$residuals~dat.advertising$TV,pch=16,col='red',main='Residual plot (original)
abline(h=0,lwd=3)

plot(fit.ylog$residuals~dat.advertising$TV,pch=16,col='red',main='Residual plot (log)',y
abline(h=0,lwd=3)


plot(fit.ysqrt$residuals~dat.advertising$TV,pch=16,col='red',main='Residual plot (sqrt)'
abline(h=0,lwd=3)
```

6.6 Influential Observations and Outliers

The influential observations are samples that has a large leverage. You should search for a formal definition of leverage if you are interested. In plain words, the inflential observations are data points that live far away from others in terms of their values of the covariates.

Outliters are the observations whose responses are far away from observations with similar covariates. We can see these from the usual scatter plots. We will not cover formal testing or measures for influential observations and outliers in this class.

```
influential_cov=dat.advertising$TV;influential_cov[1]=500;
outliers_resp=fit.lm$residuals;outliers_resp[1]=30;
```

```r
par(mfrow=c(1,2))
plot(fit.lm$residuals~influential_cov,pch=16,col='red',main='Influential obs.',xlab='TV'
abline(h=0,lwd=3)
points(y=fit.lm$residuals[1],x=influential_cov[1],col='green',pch=16)


plot(outliers_resp~dat.advertising$TV,pch=16,col='red',main='Outlier',xlab='TV')
abline(h=0,lwd=3)
points(y=outliers_resp[1],x=dat.advertising$TV[1],col='green',pch=16)
```

Chapter 7

Multiple covariates

7.1 Examples

Reading materials: Slides 101 - 110 in STA108_LinearRegression_S20.pdf.

We will revisit the examples that have been studied in Chapter 1.

7.1.1 Advertising data

We now consider all the covariates in the data set. The resulting model is

$$y_i = \beta_0 + \sum_{j=1}^{3} x_{i,j}\beta_j + \epsilon_i, i = 1, \ldots, 200,$$

where y_i is the `sales` (1 unit = 1000 dollars) for the ith entry, $x_{i,1}$ is the TV advertising budget for the ith entry, $x_{i,2}$ is the radio advertising budget for the ith entry, $x_{i,3}$ is the newspaper advertising budget for the ith entry. In addition, we assume that the errors $\{\epsilon_i\}_{i=1}^{200}$ satisfy that $\epsilon_1, \ldots, \epsilon_200$ are independently and identically distributed (i.i.d.), $\mathbb{E}[\epsilon_i] = 0$ for $i = 1, 2, \ldots, 200$ and $\text{var}(\epsilon_i) = \sigma^2$ for $i = 1, 2, \ldots, 200$. Recall that we consider fixed design (i.e., x_i is not random) in this course for simplicity.

```
dat.advertising=read.csv('./data/advertising.csv');
# Fit a multiple linear regression
fit.advertising = lm(sales~TV+radio+newspaper+1,data=dat.advertising);
# Summarize the fitted results
summary(fit.advertising)

fit.advertising.slr = lm(sales~TV+1,data=dat.advertising);
# Summarize the fitted results
summary(fit.advertising.slr)
```

How would you interpret the fitted results? How do they compare with the results from the simple linear regression in Chapter **??**?

7.1.2 Flu shot

We will include `age` and `female`(gender) in the regression in addition to the treatment received. What can you conclude from the fitted results?

```
dat.flu = read.table("./data/flu240.txt", header = TRUE)
# Fit a multiple linear regression
fit.flu= lm(outcome~treatment.received+age+female+1,data=dat.flu);
# Summarize the fitted results
summary(fit.flu)
```

In addition, we can fit a regression with treatment received as the outcome, where treatment assigned, age and gender are covariates. What can you conclude now?

```
fit.trt= lm(treatment.received~treatment.assigned+age+female+1,data=dat.flu);
# Summarize the fitted results
summary(fit.trt)
```

7.1.3 Project STAR

Project STAR is a stratified randomized experiment, where randomization happened within each school. Hence, we will add `schoolid2` as an additional factor in the regression.

```
library(AER)
data("STAR")
dat.STAR=STAR; # Just to be consistent
# Fit a simple linear regression
fit.STAR= lm(math2~as.factor(star2)+as.factor(schoolid2)+1,data=dat.STAR);
# Summarize the fitted results
summary(fit.STAR)$coef[1:3,]

# We can no longer draw a fitted line here...
```

7.2 Classification of variables

Reading materials: Slides 111 - 118 in STA108_LinearRegression_S20.pdf.

We will use simulation to examine the roles of the variables.

```
source('sources.r')

simulate.one.instance.data<-function(){
n=100;
confounder = rnorm(n)*2;
precision = runif(n);
instrument = rbinom(n,size=1,prob=0.4);
trt=rnorm(n) + confounder*0.4-instrument;
```

```
y=trt*1-0.5*confounder+precision*0.7+0.5*rnorm(n);
out=data.frame(y=y,trt=trt,confounder=confounder, precision=precision, instrument=instru
return(out)
}
```

```
## Effect if ignoring confounder:

simulate.one.instance<-function(){
  dat=simulate.one.instance.data();
  beta.hat=fit.linear.model(covariate=dat[,'trt'],outcome=dat$y)

  beta.hat.confounder=fit.linear.model(covariate=as.matrix(dat[,c('trt','confounder')])
  return(c(beta.hat[2],beta.hat.confounder[2]))
}
N.sim=10000;

set.seed(1)

sim.confounder=replicate(N.sim, simulate.one.instance());
apply(sim.confounder,1,mean) # The true value is 1
apply( (sim.confounder-1),1,function(x){ sum(x^2) }) # The mean squared error
```

```
## Effect if ignoring precision variable:

simulate.one.instance<-function(){
  dat=simulate.one.instance.data();
  beta.hat=fit.linear.model(covariate=as.matrix(dat[,c('trt','confounder')]),outcome=da

  beta.hat.precision=fit.linear.model(covariate=as.matrix(dat[,c('trt','confounder','pr
  return(c(beta.hat[2],beta.hat.precision[2]))
}
N.sim=10000;

set.seed(1)

sim.precision=replicate(N.sim, simulate.one.instance());
apply(sim.precision,1,mean) # The true value is 1
apply( (sim.precision-1),1,function(x){ sum(x^2) }) # The mean squared error
```

```
## The power of instruments
###  If the confounder is unobserved, we can use the two stage least squares method

simulate.one.instance<-function(){
  dat=simulate.one.instance.data();
  beta.hat.naive=fit.linear.model(covariate=as.matrix(dat[,'trt']),outcome=dat$y)
```

```
  yiv=fit.linear.model(covariate=as.matrix(dat[,c('instrument')]),outcome=dat$y)

  trtiv=fit.linear.model(covariate=as.matrix(dat[,c('instrument')]),outcome=dat$trt)


  return(c(beta.hat.naive[2],yiv[2]/trtiv[2]))
}
N.sim=10000;

set.seed(1)

sim.iv=replicate(N.sim, simulate.one.instance());
apply(sim.iv,1,mean) # The true value is 1
apply( (sim.iv-1),1,function(x){ sum(x^2) }) # The mean squared error
```

7.3 Least squares estimation

Reading materials: Slides 119 - 141 in STA108_LinearRegression_S20.pdf.

```
# Consider another data set with a bit more covariates
library(AER)
data("Fatalities")
dat.fatalities = Fatalities
# This dataset contains 34 variables
# It is actually a longitudinal/panel data
# We will ignore the spatial temporal structure in this class

y=dat.fatalities$fatal; #Number of vehicle fatalities.
X=as.matrix(dat.fatalities[,c('spirits','unemp','income','beertax','baptist','mormon','c


# Rewrite the functions in Chapter 2
linear.model<-function(beta,covariate){
  # beta: a 2-vector, where the first entry is the intercept
  beta=as.matrix(beta,nrow=length(beta))
  yout=covariate%*%beta[-1]+beta[1];
  return(yout);
  # Note: this function only works with simple linear regression model
  # How would you generalize it?
}
sum.of.squares<-function(beta,covariate,outcome){
  yout=linear.model(beta=beta,covariate=covariate);
  res=outcome-yout;
```

```
  sos= sum(res^2);
  return(sos)
}
fit.linear.model<-function(covariate,outcome){
  X=cbind(1,covariate);

  beta.fit=solve( t(X)%*%X )%*%t(X)%*%outcome;
  return(beta.fit)
}

beta.hat=fit.linear.model(covariate=X,outcome=y)

# Compare with lm()
fit.lm=lm(y~X+1);
beta.hat
```

```
# Hat matrix
calculate.hat<-function(covariate){
  X=cbind(1,covariate);
  P=X%*%solve(t(X)%*%X)%*%t(X);
  return(P)
}
P=calculate.hat(covariate=X);

# Verify the properties of the hat matrix:
## Residuals
resid= y-linear.model(beta=beta.hat,covariate=X);
resid.hat = (diag(length(y))-P)%*%y;
sum((resid-resid.hat)^2)

## Idempotent
max( abs(P%*%P-P))

## Orthogonality
max(abs((diag(length(y))-P)%*%X))
```

```
# Sum of squares with the hat matrix:
n=length(y);J=matrix(1,n,n);I=diag(n)
total.sum.of.squares= t(y)%*%(I-J/n)%*%y
explained.sum.of.squares=t(y)%*%(P-J/n)%*%y
residual.sum.of.squares=t(y)%*%(I-P)%*%y

# Check if it is true
sum.of.squares(beta.hat,covariate = X,outcome = y)
```

```
# R^2
R.sq = explained.sum.of.squares/total.sum.of.squares;

# Through more variables into X
# These variables are meaningless
set.seed(1)
Z=matrix(rnorm(5*n),nrow=n,ncol=5);
X.Z=cbind(X,Z);

P.Z=calculate.hat(covariate=X.Z);

R.sq.Z= t(y)%*%(P.Z-J/n)%*%y/t(y)%*%(I-J/n)%*%y;

R.sq.Z
R.sq

# Hence the adjusted R^2:
R.sq.adj=1-(residual.sum.of.squares/(n-dim(X)[2]-1) )/(total.sum.of.squares/(n-1));

R.sq.Z.adj=1-(t(y)%*%(I-P.Z)%*%y/(n-dim(X.Z)[2]-1) )/(t(y)%*%(I-J/n)%*%y/(n-1));
```

7.4 Underfitting and overfitting

Reading materials: Slides 142 - 152 in STA108_LinearRegression_S20.pdf.

```
## Underfitting
simulate.one.instance<-function(beta.true,X,Z){
  eta=c(3,4);
 n= dim(X)[1];
  noise = rnorm(n)/2;
  y.XZ=X%*%beta.true+Z%*%eta + noise;

  beta.full = fit.linear.model(covariate=cbind(X,Z),outcome=y.XZ)
  beta.under= fit.linear.model(covariate=X,outcome=y.XZ)

  return(cbind(beta.full[2:3]-beta.true,beta.under[2:3]-beta.true))
}

beta.true=c(1,2)
n=100;
X=matrix(rnorm(2*n),ncol=2)
Z=matrix(rnorm(2*n),ncol=2)

set.seed(1);
```

```
N.sim=10000;
under.sim=replicate(N.sim,simulate.one.instance(beta.true,X,Z))
# Fitting the true model gives

apply(under.sim[1,,],1,mean) # Non-zero bias

# Does it contradict the claim about precision variable?
# Try another simulation:
simulate.one.instance.pre<-function(beta.true){
  eta=c(3,4);
n=100;
X=matrix(rnorm(2*n),ncol=2)
Z=matrix(rnorm(2*n),ncol=2)
  noise = rnorm(n)/2;
  y.XZ=X%*%beta.true+Z%*%eta + noise;

  beta.full = fit.linear.model(covariate=cbind(X,Z),outcome=y.XZ)
  beta.under= fit.linear.model(covariate=X,outcome=y.XZ)

  return(cbind(beta.full[2:3]-beta.true,beta.under[2:3]-beta.true))
}

beta.true=c(1,2)

set.seed(1);
N.sim=10000;
pre.sim=replicate(N.sim,simulate.one.instance.pre(beta.true))

apply(pre.sim[1,,],1,mean) # almost zero bias
```

```
### Overfitting
# Just modify the code slightly..
simulate.one.instance<-function(beta.true,X,Z){
  eta=c(3,4);
 n= dim(X)[1];
  noise = rnorm(n)/2;
  y.X=X%*%beta.true+ noise;

  beta.over= fit.linear.model(covariate=cbind(X,Z),outcome=y.X)
  beta.full= fit.linear.model(covariate=X,outcome=y.X)

  return(cbind(beta.full[2:3]-beta.true,beta.over[2:3]-beta.true))
}
```

```
beta.true=c(1,2)
n=100;
X=matrix(rnorm(2*n),ncol=2)
Z=matrix(rnorm(2*n),ncol=2)

set.seed(1);
N.sim=10000;
over.sim=replicate(N.sim,simulate.one.instance(beta.true,X,Z))
# Fitting the true model gives

apply(over.sim[1,,],1,mean) # No bias
```

7.5 Sampling distribution and inference

Reading materials: Slides 153 - 165 in STA108_LinearRegression_S20.pdf.

We will skip the part for multivariate normal distribution. You will learn more about this in STA 135: Multivariate Data Analysis.

7.5.1 ANOVA

```
attach(dat.advertising)
full.model=lm(sales~TV+radio+newspaper);
reduced.model=lm(sales~TV);
anova(reduced.model,full.model)

RSS.Reduced<-sum((summary(reduced.model)$residuals)^2);
RSS.Full<-sum((summary(full.model)$residuals)^2);
df.Reduced <- summary(reduced.model)$df[2];
df.Full <- summary(full.model)$df[2];

tF<-((RSS.Reduced-RSS.Full)/(df.Reduced-df.Full))/(RSS.Full/df.Full);
tF
```

7.5.2 Wald test

Suppose that we want to test the null hypothesis

$$H_0 : \beta_1 - 2\beta_2 = 0 \text{ v.s. } H_a : \beta_1 - 2\beta_2 \neq 0,$$

where β_1 is the regression coefficient for `TV` and β_2 is the regression coefficient for `radio`.

```
full.model=lm(sales~TV+radio+newspaper);
covariance.LSE=vcov(full.model); # Calculate the covariance for the least squares estim
```

```
coef.LSE=coef(full.model);
covariance.test = covariance.LSE[2:3,2:3]; # Extract the submatrix of covariance for be
coef.test=coef.LSE[2:3];
R=matrix(0,1,2);
R[1,1]=1;R[1,2]=-2;

test.stat=t(R%*%coef.test)%*%solve(R%*%covariance.test%*%t(R))%*%(R%*%coef.test);
p.value=1-pchisq(test.stat,df=1);
```

We can verify that the Wald test yields similar results with the F-test using `anova`.

$$H_0 : \beta_1 = \beta_2 = 0 \text{ v.s. } H_a : \beta_1 \neq 0, \beta_2 \neq 0.$$

```
full.model=lm(sales~TV+radio+newspaper);
covariance.LSE=vcov(full.model); # Calculate the covariance for the least squares estim
coef.LSE=coef(full.model);
covariance.test = covariance.LSE[2:3,2:3]; # Extract the submatrix of covariance for be
coef.test=coef.LSE[2:3];
R=matrix(0,2,2);
R[1,1]=1;R[2,2]=1;

test.stat=t(R%*%coef.test)%*%solve(R%*%covariance.test%*%t(R))%*%(R%*%coef.test);
p_value=1-pchisq(test.stat,df=2);


reduced_model=lm(sales~newspaper);
anova(reduced.model,full.model)
```

Chapter 8

Model selection

```
library(AER)
data("Fatalities")
dat.fatalities = Fatalities[,c('fatal','spirits','unemp','income','beertax','baptist','m
```

8.1 Criteria

Reading materials: Slides 166 - 180 in STA108_LinearRegression_S20.pdf.

```
# Information criteria


fit.full=lm(fatal~spirits+unemp+income+beertax+baptist+mormon+drinkage+dry+youngdrivers+
fit.six=lm(fatal~spirits+unemp+income+beertax+baptist+mormon,data=dat.fatalities)

BIC(fit.full)
BIC(fit.six)


AIC(fit.full)
AIC(fit.six)
```

```
# cross-validation:
# We will use a package for this:
library(boot)


fit.full.g=glm(fatal~spirits+unemp+income+beertax+baptist+mormon+drinkage+dry+youngdrive
fit.six.g=glm(fatal~spirits+unemp+income+beertax+baptist+mormon,data=dat.fatalities)
```

```
set.seed(1)
cv.full.10=cv.glm(dat.fatalities,fit.full.g,K=10)
cv.full.10$delta

cv.six.10=cv.glm(dat.fatalities,fit.six.g,K=10)
cv.six.10$delta

# From the helpfile, delta is:

# A vector of length two. The first component is the raw cross-validation estimate of
```

8.2 Selection procedure

Reading materials: Slides 181 - 190 in STA108_LinearRegression_S20.pdf.

```
# Model selection procedure

# Forward selection & AIC
step(lm(fatal~spirits+unemp+income+beertax+baptist+mormon+drinkage+dry+youngdrivers+mile


# Backward selection & AIC
step(lm(fatal~spirits+unemp+income+beertax+baptist+mormon+drinkage+dry+youngdrivers+mile



# Backward selection & BIC
step(lm(fatal~spirits+unemp+income+beertax+baptist+mormon+drinkage+dry+youngdrivers+mile
```

```
# Penalized regression: lasso
library(glmnet)

y=dat.fatalities$fatal; #Number of vehicle fatalities.
X=as.matrix(dat.fatalities[,c('spirits','unemp','income','beertax','baptist','mormon','d


fit.cv.glmnet=cv.glmnet(y=y,x=X,nfold=10,family='gaussian');

fit.cv.glmnet$lambda.min # selected tuning parameter

fit.glmnet=glmnet(x=X,y=y,family='gaussian',alpha=1,lambda=fit.cv.glmnet$lambda.min)

fit.glmnet$beta # One of the fitted coefficient is zero
```

Appendix A

R basics

Reading materials: Sections 4, 6, and 8 in R for data science by Garrett Grolemund and Hadley Wickham (optional).

This section reviews some basic functions in R that will be useful in this class.

A.1 Basic objects in R

```
## Generating a vector with seq()

seq(from = 3.1, to = 8.9, by = 0.8)
seq(from = 3.1, to = 8.9, length.out = 10)
# seq(from = 3.1, to = 8.9, by=1, length.out = 10)

### checking for equality of two vectors
a=1:3
b=c(1,2,3) #b<-c(1,2,3)
a == b

### How do we check if *all* entries are the same?
all(a==b)

### How about the identical function?
identical(a,b)
typeof(a)
typeof(b)

### Logical and character vectors
u = c(T,F,T,F,F,T)
typeof(u)
```

```
### Define character vectors
w=c('a', 'b','c','e');

### What happens when merging character vectors with numbers?
y=c(1:3,w)
typeof(y[1])
y=c(pi,w)

### What if we force them to be numeric?
#?as.numeric
as.numeric(y[1])
as.numeric(y)

### Searching for elements of a vector
### Using > == < & |, and which
x = c(1,3,4,6.7,-4,NA,-2.9)

x
x==1
x> 1
x< 1
x >= 1
x<=1
x<0
x[5]

x[which(x<0)]
x[x<0]


### Filtering of vectors
# 0< x < 2
x[(0 < x) & (x < 2)]
which((0 < x) & (x < 2))

x[ !(x<0)]
!(x<0)
x[-5]
```

A.2 Summary statistics

```
x = c(1,3,4,6.7,-4,NA,-2.9)
x
```

```
length(x) # you have seen this already
sum(x) # summation of all elements of x
sd(x) # standard deviation
### Why are they all NA?

### working with NA (missing values), NULL, and NaN
y=NULL;
0/0 # NaN

1/0
-1/0
### How would you find the NAs in x?
x==NA
which(is.na(x))
is.null(y)
#is.nan()
#is.infinite()


sum(x,na.rm=TRUE)
# ?sum
args(sum) # check the default argument(s) for sum()
example(sum)
### Summarize  matrices using apply()
```

A.3 Data structures

A.3.1 String

```
wrd = c("a","new","sentence")
u = c("_","!","?")
wrd
typeof(wrd)

print("this is a sentence")
length("this is a sentence")
length(wrd)

### Use paste() function to manipulate strings
paste(wrd,u,sep="/")

### Use strsplit() to split strings
```

```
split.str=strsplit("this is a sentence", split=" ")

typeof(split.str)
length(split.str)
split.str[[1]]
```

A.3.2 Data frame

```
### manipulation of data frame with base R
### No dplyr
### creating and accessing data frame
name =c("Bob","Jane")
age = c(19,21)
GPA = c(3.2,3.6)
students = data.frame(name,age,GPA,stringsAsFactors=F)
students
str(students)


### Add new entry into the data frame
students = rbind(students,c("Ashley",20,3.4))

### Add new variable:
students = cbind(students,resident=c(T,F,T))

students
##-------------------------------------##
```

A.4 List

```
## List ####
bob=list(name="bob",age=19,school="UCD",GPA=3.2,resident=T)

### Use lapply() with list
newlist = list(field_one= runif(10), field_two=runif(20),field_three=runif(50) );
lapply(newlist,sum)
##-------------------------------------##
```

A.5 Functions in R

```
### Evaluating a function
x = seq(0,1,0.05)
x
y = sin(2*pi*x)*exp(-4*x^2)

### Writing a function
sinexp = function(z){
  y=sin(2*pi*z)*exp(-4*z^2);
  return(y)
}

sinexp(x)
w=sinexp(x)

y==w
all(y==w)
identical(y,w)
```

A.6 Miscellaneous

```
### Checking the state of work session
objects()  # all the objects in the current R session
getwd()  # print the current working directory
remove()
#setwd()

#?persp
example(persp)

#data()
```

Appendix B

Linear algebra

Reading materials: STA108_LinearAlgebra_S20.pdf.

B.1 Vector

```
x = c(1,2,4,2.5,-1.7,1) # creates a vector (you can use "<-" instead of "=")

length(x) # check the length of x
class(x) # class of the object; you can also check typeof()
### Try z=list(); class(z);

x[ 3 ] # extract one element in x..
x[c(1,2,3)]
x[2:4]

z<-numeric(10)

### Transpose:
x.t=t(x);
length(x.t)
x.t
x

y = x^2
### Q: what is the length of y?
length(y)

### The "default" operation in R is elementwise:
x/y # elementwise division
y-x*x # all elementwise..
```

```
  ### There are other operators..
  x %o% y # outer product of x and y

  x %*% y # Matrix product. Q: what is R doing here?

  sum(x*y)
```

B.2 Matrix

```
M = matrix(0,2,3) # matrix with 2 rows and 3 columns with all entries 0
# ?matrix


Xmat1 = matrix(x,2,3) # 2 x 3 matrix formed by elements of vector x,
### ordered in *column-wise* manner
Xmat2 = matrix(x,2,3,byrow=T) # ordered row-wise


M<- Xmat1;
dimnames(M)
rownames(M) = c("first","second")
colnames(M) = c("a","b","c")

dimnames(M)
typeof(dimnames(M))

### access elements through names of rows/columns
M[,"a"]
M[,1]

M[,c("a","c")]
M[,c(1,3)]

M %*% c(4,-3,1)  # multiply matrix M to vector c(4,-3,1)
M + c(1,-2) # add the column vector c(1,-2) to each column of M

M2 = cbind(M,c(4,5))  # create matrix M2 by appending column c(4,5) to M
### How to append row to M2?
new_row = c(1,3,5,7)
#?cbind
#?rbind
```

```
### What if we call length() on a matrix?
length(M)
dim(M)




### Transpose of the matrix
M.t=t(M);
identical(t(M.t),M)

### Diagonal and identity matrix

## Identity matrix
diag(5)
## Diagonal matrix
diag(c(1,2,3,4,5))
## Symmatric matrix
A= matrix(rnorm(9),3,3);
A=A+t(A); # To make sure it is symmetric
identical(A,t(A))


### Trace of a matrix:
sum(diag(A))
## Define a tr() function to match the common convention
tr<-function(x){
  out=sum(diag(x));
  return(out)
}
tr(A)

## Verify some properties of the trace
B= matrix(rnorm(9),3,3);
tr(A)
tr(t(A))

tr(A+B)
tr(A)+tr(B)

tr(A%*%B)
tr(B%*%A) # if the dimensions check out..

tr(A)
```

```r
sum(eigen(A)$values)

### Matrix-vector product:

Xmat1 = matrix(rnorm(6),2,3)
xvec1=rnorm(3)
Xmat1%*%xvec1

### elementwise product of Xmat1 and Xmat2
Xmat1*Xmat2



### Matrix multiplication
# Xmat1%*%Xmat2
dim(Xmat1)
dim(Xmat2)
Xmat1%*%t(Xmat2)
### Verify another property of transpose:
t( Xmat1%*%t(Xmat2) )
Xmat2%*%t(Xmat1)
# solve(Xmat1) # what does the error message say?
```

```r
### Inverse of a marix
# set.seed(1)
# A= matrix(rnorm(9),3,3);
# set.seed(): specify the random seed for generating random numbers
# Because a computer (if not a quantum one) can only generate pseudorandomness
# set.seed(1)
# B=matrix(rnorm(9),3,3);
# identical(A,B)

set.seed(1)
A= matrix(rnorm(9),3,3);
A.inv=solve(A);

A.inv%*%A
A%*%A.inv


B=matrix(rnorm(9),3,3);
identical(A,B)
B.inv=solve(B);
```

```r
AB.inv=solve(A%*%B)

B.inv%*%A.inv
AB.inv

max(abs(B.inv%*%A.inv-AB.inv))

# Why?
# This is because
# (A%*%B)%*%AB.int= I if AB.inv is the inverse of AB=A%*%B
# and that
# A%*%B %*%B.inv%*%A.inv=A%*%(B%*%B.inv)%*%A.inv =A%*%A.inv=I

### Projection matrix

set.seed(12)
X = matrix(rnorm(6),3,2)
P=X%*%solve(t(X)%*%X)%*%t(X)

P%*%X
X
(diag(3)-P)%*%X

### Rank of a matrix:
qr(X)$rank # Check out the qr() function yourself!
# The rank of X is 2 when the dimension of X is 3 by 2
# This means that the matrix X is full rank


## How about x.transpose times X?
XtX=t(X)%*%X
dim(XtX)
qr(XtX)$rank
# This is a full rank matrix

## How about X times X.transpose?
XXt=X%*%t(X)
dim(XXt)
qr(XXt)$rank
# This is NOT a full rank matrix

# Rank and inverse
solve(XtX)
# solve(XXt)
```

```
### Determinants
det(A)
1/det(solve(A))
det(A%*%B)
det(A)%*%det(B)
det(3*A)

A.symm = (A+t(A))/2;
det(A.symm)
A.symm.eigenvalues=eigen(A.symm)$values
prod(A.symm.eigenvalues)
# Trace: summation of eigenvalues
# Determinant: product of eigenvalues (for symmetric matrix only!)


### Eigenvalue and eigenvector
A.symm.eigenvectors=eigen(A.symm)$vectors

A.symm%*%A.symm.eigenvectors[,1]
A.symm.eigenvalues[1]*A.symm.eigenvectors[,1]
A.symm.eigenvalues
qr(A.symm)$rank

qr(XXt)$rank
eigen(XXt) # the last eigenvalue equals zero, with some numerical error



### Matrix definiteness
eigen(XtX)
eigen(P)


### Eigenvalue decomposition
A.symm.eigenvectors%*%diag(A.symm.eigenvalues)%*%t(A.symm.eigenvectors)
A.symm


### Singular value decomposition
svd(X)  # singular value decomposition of X
```

B.3 Other operations on vectors and matrices

```
### How does R handle mathematically invalid operations?
### Example 1:
 x = c(1,2,4,2.5,-1.7,1)
w =1:3  # short-handed for c(1,2,3)
### Q: What is the length of w?

x + w  # no error displayed, but not quite a mathematically valid operation
### Q: what did R just do???
c(x[1:3]+w, x[4:6]+w)

### What if...
# Create a vector of length 4 (from 1 to 4)
n<-1:4;
x+n

### Example 2:

z = sin(x) + x^{1.5}* exp(-x^3)
### What is (-1.7)^{1.5}???
### How did R handle this?
x[5]
z[5]
```

B.3.1 Array

```
### Arrays, a.k.a. tensors
### A matrix is a two-way array

# ?array # check for R documentation on array

A = array(rnorm(2*3*4),dim=c(2,3,4))  # creates an array with entries that are i.i.d. l
#?rnorm
dim(A) # checks the dimension of A

A[,1,] # returns the 2 x 4 matrix formed by restricting the second dimension index to

### What is the dimension of A[,1,]?

A[2,,] # returns the 3 x 4 matrix formed by restricting the first dimension index to 2

A[2,1,2]
A[A<0]
```

B.3.2 Datasets as matrices

```
library(AER) # install.packages('AER')
data("Fatalities")
#?Fatalities

data("CreditCard")
dat.CC=CreditCard
#?CreditCard
dim(dat.CC)
head(dat.CC)

Y.CC=dat.CC[,1]=='yes';
X.CC=as.matrix(dat.CC[,c('reports','age','income','share','expenditure','dependents','mo

# Check the rank of the covariates:
qr(X.CC)$rank
dim(X.CC) # full rank, i.e., no multicolinearity


# Linear regression:
fit.CC=lm(Y.CC~X.CC-1);
summary(fit.CC)

# Fitted coefficients:
beta.hat=solve(t(X.CC)%*%X.CC)%*%t(X.CC)%*%Y.CC;
fit.CC$coefficients-beta.hat

# Hat matrix:
P.CC=X.CC%*%solve(t(X.CC)%*%X.CC)%*%t(X.CC)
# Fitted Y
Yhat.CC=P.CC%*%Y.CC
# Compare with the fits from lm()
max(abs(fit.CC$fitted.values-Yhat.CC))
```

Appendix C

Simulation and visualization

Reading materials: Chapters 3, 5, and 7 in R for data science by Garrett Grolemund and Hadley Wickham (optional).

C.1 Simulation and visualization: univariate

```
## Random seed
set.seed(1) # set random number generator seed for reproducibility
runif(1)
runif(1)

set.seed(2)
runif(1)

set.seed(1)
runif(1)
runif(1)
```

```
## Uniform distribution
### Draw ten uniform random variables:
runif(10)
# use ?runif to see what other options are available

# To help understand this function, we can draw the density plot for this distribution
x.grid=seq(from=0,to=1,length.out=100)
unif.pdf=dunif(x.grid,min=0,max=1);
plot(y=unif.pdf,x=x.grid,type='l')

# Compare this with a histogram from a set of random variables from runif:

unif.rv.100=runif(100);
```

```
hist(unif.rv.100,freq=FALSE,main='Uniform',xlab='X')
lines(y=unif.pdf,x=x.grid,col='blue',lwd=3)
# Further compare it with a smooth density estimation based on the samples:
unif.pdf.est=density(unif.rv.100);
lines(unif.pdf.est,col='red',lwd=3)
```

```
# Wrap up the above plotting code as a function

plot.density.empirical<-function(rvs,pdf.true,pdf.grid,main){
hist(rvs,freq=FALSE,main=main,xlab='X')
lines(y=pdf.true,x=pdf.grid,col='blue',lwd=3)
# Further compare it with a smooth density estimation based on the samples:
pdf.est=density(rvs);
lines(pdf.est,col='red',lwd=3)
}


plot.density.empirical(unif.rv.100,unif.pdf,x.grid,main='Uniform')
```

```
### Normal
### Draw ten normal random variables with mean 0 and variance 2
rnorm(10,mean=0,sd=sqrt(2))

### There are many other distributions in R
### try ?rchisq, ?rf, ?rt, ?rbeta, ?rpois, ...
```

```
## Other distributions
### Poisson  ?rpois

### Exponential ?rexp

### t ?rt


### Chisq ?rchisq


### F ?rf
```

```
## Sampling from a user-specified vector

### Draw samples from any vectors using sample()
wrd = c("yet", "a","new","sentence")
sample(wrd,size=2)
# use ?sample to see other options
```

```
### Visulize the simulated data

y<-rnorm(100);
# We start with the very basic histogram
hist(y)
# Use ?hist to modify the plot

# And the boxplot:
boxplot(y)

# We can also draw visualize the data using ggplot2
library(ggplot2)
dat=data.frame(y=y)
ggplot(dat, aes(x=1,y=y)) +
  geom_violin(trim=FALSE)

# Read more about the violin plots in the post here:
# http://www.sthda.com/english/wiki/ggplot2-violin-plot-quick-start-guide-r-software-a
```

C.2 Simulation and visualization: multivariate

```
# We can generate many independent univariate random variables using code in the previ

# Here we will generate dependent random variables, for instance,
n=50;# sample size
x1=rnorm(n);
x2=x1*2+runif(n)*2;
x3=x1*x2*rpois(n,lambda=3);

# You can also generate a random variable using its probability density function using
# We will be fine with the simple data generating method in this class

# We put the three random variables into one data.frame for ease of plotting
dat.multi<-data.frame(x1=x1,x2=x2,x3=x3)
```

```
### Visualization

### Pairwise scatter plot
pairs(dat.multi,pch='.',cex=4)


### Visualization in ggplot2
library(GGally)
```

```r
ggpairs(dat.multi)


### For the pair x1 and x3

# A more informative scatterplot, using base R
plot(x=dat.multi$x1,y=dat.multi$x3,data=dat.multi,pch=16,cex=2)
lines(lowess(dat.multi$x1, dat.multi$x3), col=2,lwd=3)


# Or using ggplot2
ggplot(data = dat.multi) +
  geom_point(mapping = aes(x = x1, y = x3)) +
  geom_smooth(mapping = aes(x = x1, y = x3))
```