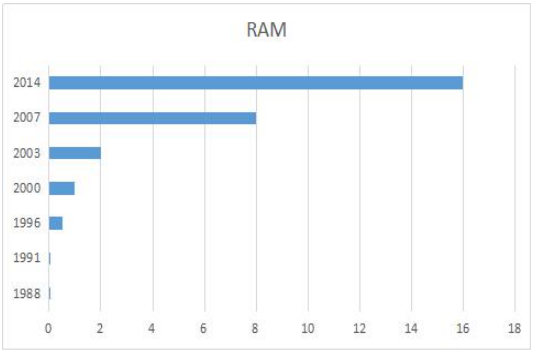
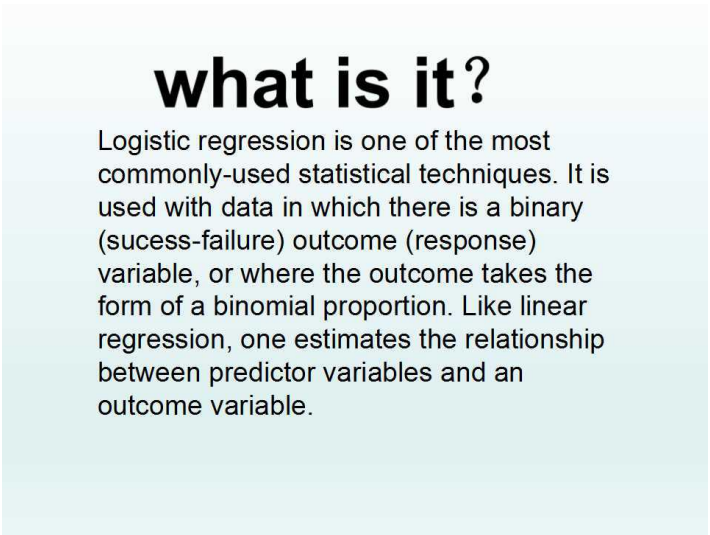
HW1



The memory of PCs over the last 30 years keep increasing all the time. In the initial years, it rise slowly, but after 2003, it increases explosively.

2.



function

Using logistic regression to predict class probabilities is a modeling choice, just like it’ s a modeling choice to predict quantitative variables with linear regression.

likehood

For each training data-point, we have a vector of features, xi , and an observed class, yi . The probability of that class was either p, if yi = 1, or 1 − p, if yi = 0. The likelihood is then

$$L(\beta_0, \beta) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i)^{1-y_i}$$

the model logistic regression model

$$\log \frac{p(x)}{1-p(x)} = \beta_0 + x \cdot \beta$$

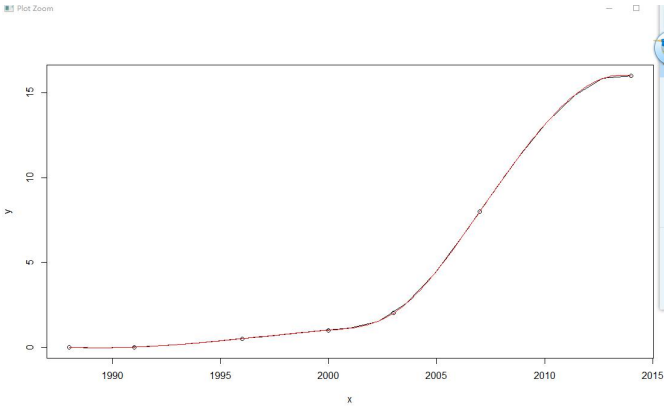
Solving for p, this gives

$$p(x; b, w) = \frac{e^{\beta_0 + x \cdot \beta}}{1 + e^{\beta_2 + x \cdot \beta}} = \frac{1}{1 + e^{-(\beta_0 + x \cdot \beta)}}$$

HW2

```
data<-read.table("C:/Users/yyy/Desktop/hw1.csv",header=TRUE,sep=",")
x<-data$Year
y<-data$RAM
plot(x,y)
```

lines(spline(x,y))
lines(spline(x, y, n = 201), col = 2)



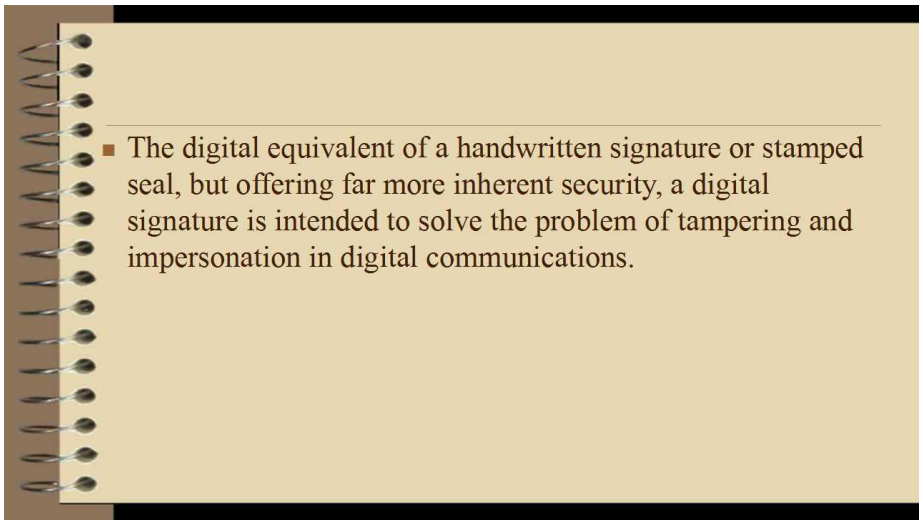
3. x = 6
n = 1000
lambda = 2
p = lambda / n
dbinom (x,2*n,p) # binomial probability mass function
dpois (x, 2*lambda) # Poisson probability mass function
dpois (0, 5)

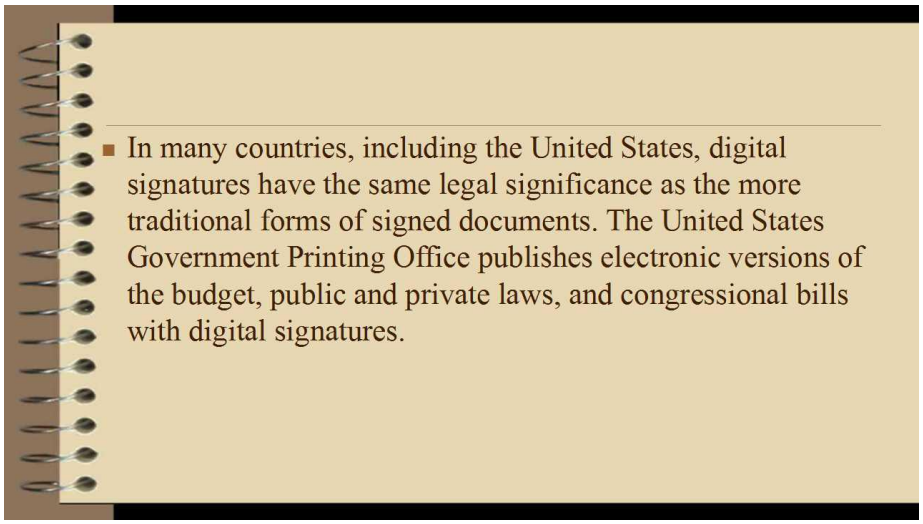
HW3
1.
> install.packages('digest')
# call the library doing the hashes
library("digest")
# now do the hash code calculation
digest("I learn a lot from this class when I am proper listening to the professor")
digest("I do not learn a lot from this class when I am absent and playing on my Iphone")

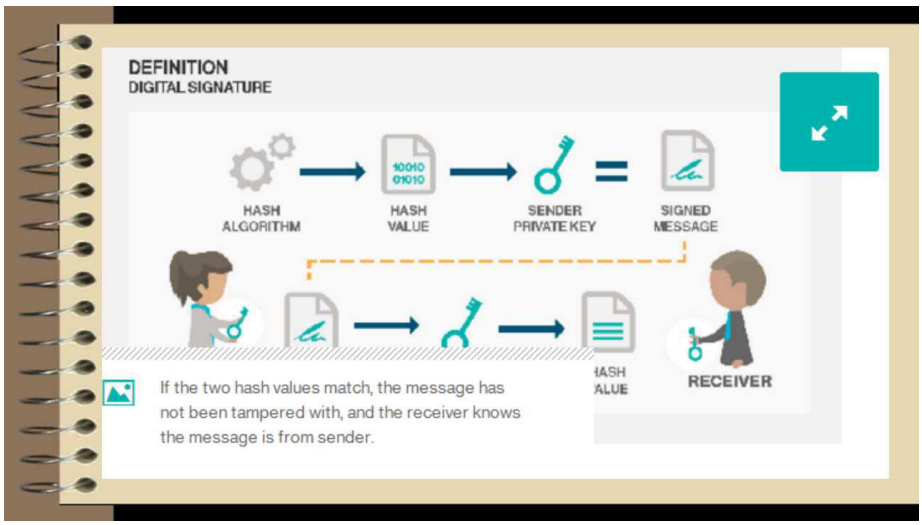
> # call the library doing the hashes> library("digest")
> # now do the hash code calculation
> digest("I learn a lot from this class when I am proper listening to the professor")
[1] "a8d3e4701672195e5dcd16ea9b062279"
> digest("I do not learn a lot from this class when I am absent and playing on my Iphone")
[1] "497edecd95aca5cc9a581e4835c3cccd"
2.

2.


The Digital Signature Algorithm (DSA)
Xinying Wu


The digital equivalent of a handwritten signature or stamped seal, but offering far more inherent security, a digital signature is intended to solve the problem of tampering and impersonation in digital communications.


In many countries, including the United States, digital signatures have the same legal significance as the more traditional forms of signed documents. The United States Government Printing Office publishes electronic versions of the budget, public and private laws, and congressional bills with digital signatures.



- a Federal Information Processing Standard for digital signatures
- proposed by the National Institute of Standards and Technology (NIST) in August 1991 for use in their Digital Signature Standard (DSS), specified in FIPS 186 in 1993.

The first part of the DSA algorithm is the public key and private key generation, which can be described as:

- Choose a prime number q, which is called the prime divisor.
- Choose another primer number p, such that p-1 mod q = 0. p is called the prime modulus.
- Choose an integer g, such that 1 < g < p, g**q mod p = 1 and g = h**((p–1)/q) mod p. q is also called g's multiplicative order modulo p.
- Choose an integer, such that 0 < x < q.
- Compute y as g**x mod p.
- Package the public key as {p,q,g,y}.
- Package the private key as {p,q,g,x}.

The second part of the DSA algorithm is the signature generation and signature verification, which can be described as:

To generate a message signature, the sender can follow these steps:

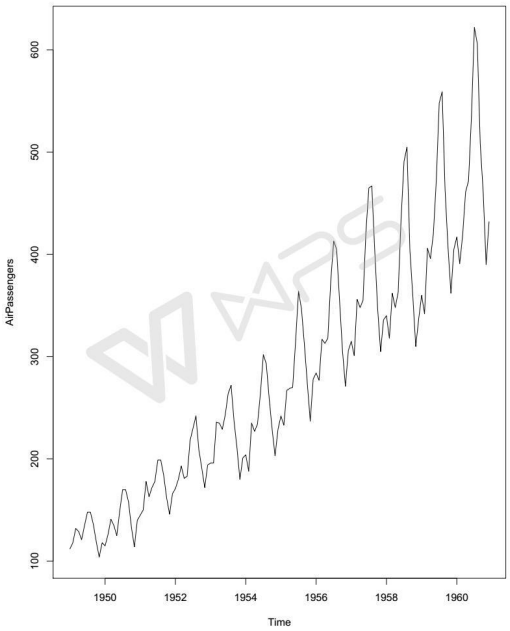
- Generate the message digest h, using a hash algorithm like SHA1.
- Generate a random number k, such that 0 < k < q.
- Compute r as (g**k mod p) mod q. If r = 0, select a different k.
- Compute i, such that k*i mod q = 1. i is called the modular multiplicative inverse of k modulo q.
- Compute s = i*(h+r*x) mod q. If s = 0, select a different k.
- Package the digital signature as {r,s}.

To verify a message signature, the receiver of the message and the digital signature can follow these steps:

- Generate the message digest h, using the same hash algorithm.
- Compute w, such that s*w mod q = 1. w is called the modular multiplicative inverse of s modulo q.
- Compute u1 = h*w mod q.
- Compute u2 = r*w mod q.
- Compute v = (((g**u1)*(y**u2)) mod p) mod q.
- If v == r, the digital signature is valid.

3.

```
# Load the package required to read JSON files.
library("rjson")
# Give the input file name to the function.
result <- fromJSON(file =
"C:/Users/yyy/Desktop/jsondata.json")
# Print the result.
print（result）
```

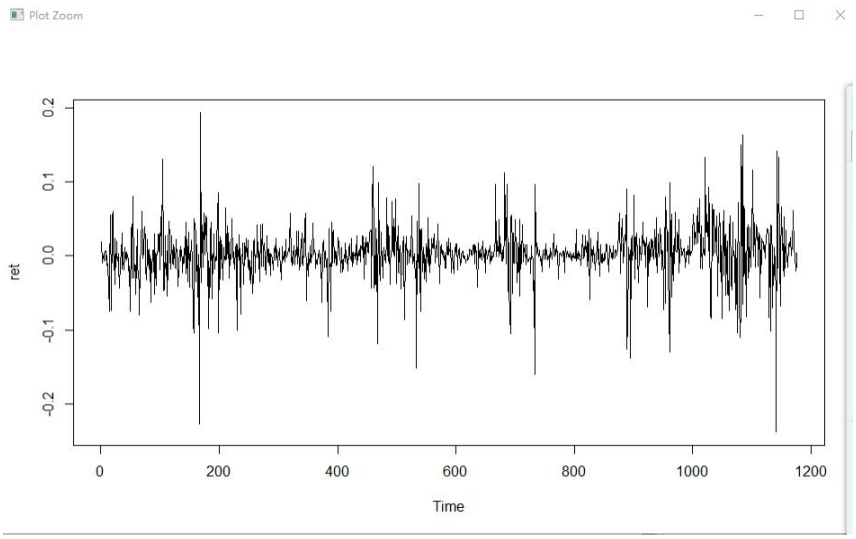


HW4

```
#install.packages("rjson", repos="http://cran.us.r-project.org")
library("rjson")
json_file = "http://crix.hu-berlin.de/data/crix.json"
json_data = fromJSON(file=json_file)
crix_data_frame = as.data.frame(json_data)
n=dim(crix_data_frame)
a=seq(1,n[2],2)
b=seq(2,n[2],2)
date=t(crix_data_frame[1,a])
price=t(crix_data_frame[1,b])

ts.plot(price)
ret=diff(log(price))
ts.plot(ret)
```

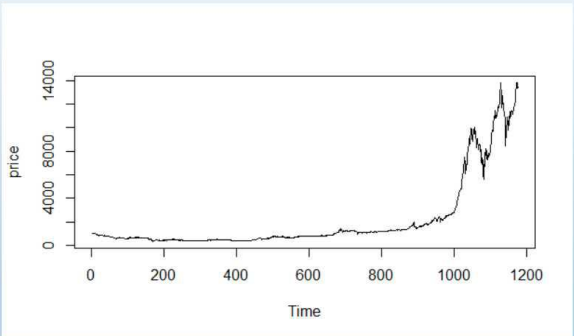
Figure4



HW4

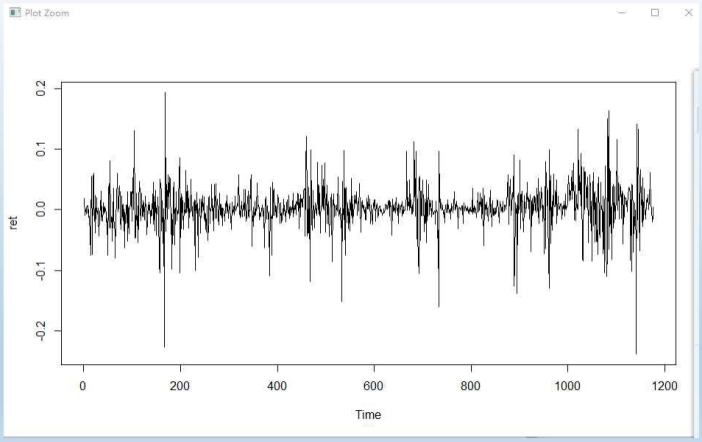
```
#install.packages("rjson", repos="http://cran.us.r-project.org")
library("rjson")
json_file = "http://crix.hu-berlin.de/data/crix.json"
json_data = fromJSON(file=json_file)
crix_data_frame = as.data.frame(json_data)
n=dim(crix_data_frame)
a=seq(1,n[2],2)
b=seq(2,n[2],2)
date=t(crix_data_frame[1,a])
price=t(crix_data_frame[1,b])

ts.plot(price)
```



```
#install.packages("rjson", repos="http://cran.us.r-project.org")
library("rjson")
json_file = "http://crix.hu-berlin.de/data/crix.json"
json_data = fromJSON(file=json_file)
crix_data_frame = as.data.frame(json_data)
n=dim(crix_data_frame)
a=seq(1,n[2],2)
b=seq(2,n[2],2)
date=t(crix_data_frame[1,a])
price=t(crix_data_frame[1,b])

ts.plot(price)
ret=diff(log(price))
ts.plot(ret)
> pdf(file="myplot.pdf")
> dev.off()
```




```
# histogram of returns
hist(ret, col = "cyan", breaks = 20, freq = FALSE, ylim = c(0, 25), xlab = NA)
lines(density(ret), lwd = 1)
mu = mean(ret)
sigma = sd(ret)
x = seq(-4, 4, length = 100)
curve(dnorm(x, mean = mean(ret), sd = sd(ret)), add = TRUE, col = "darkblue", lwd =
1)
```

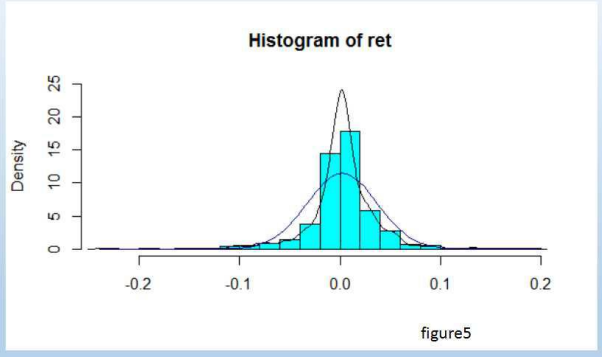
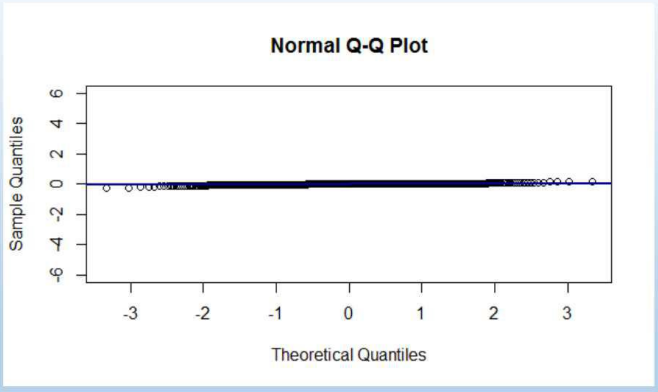


figure5

```
# let's check it by qq-plot
qqnorm(ret,ylim = c(-6, 6))
qqline(ret, col = "darkblue", lwd = 2)
```



```
# plot acf
autocorr = acf(ret, lag.max = 20, ylab = "Sample
Autocorrelation", main = NA,lwd = 1, ylim = c(-0.2,
1))
```

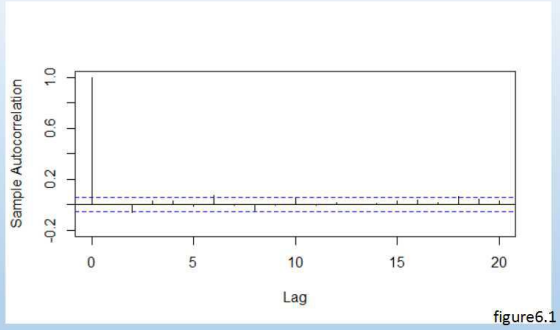


figure6.1

```
# plot of pacf
autopcorr = pacf(ret, lag.max = 20, ylab = "Sample
Partial Autocorrelation", main = NA, ylim = c(-0.3,
0.3), lwd = 2)
```

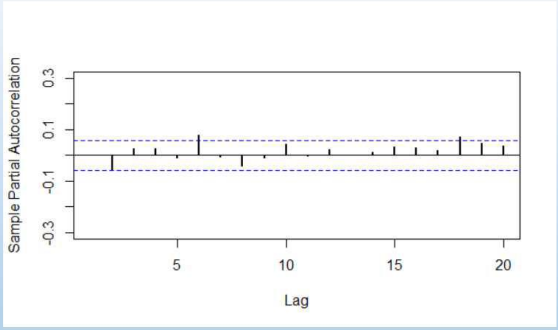
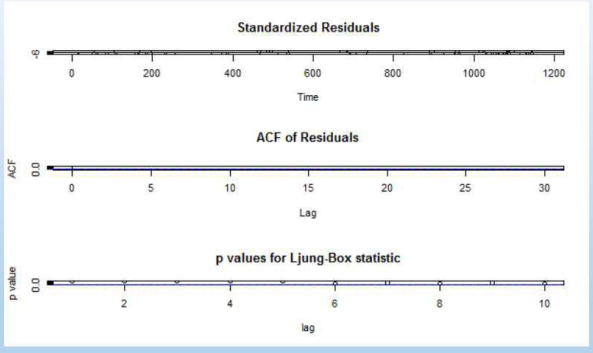


figure6.2

```
# select p and q order of ARIMA model
fit4 = arima(ret, order = c(2, 0, 3))
tsdiag(fit4)
Box.test(fit4$residuals, lag = 1)
```



```
fitr4 = arima(ret, order = c(2, 1, 3))
tsdiag(fitr4)
Box.test(fitr4$residuals, lag = 1)
> # to conclude, 202 is better than 213
> fit202 = arima(ret, order = c(2, 0, 2))
> tsdiag(fit202)
> tsdiag(fit4)
> tsdiag(fitr4)
```

```
> # arima202 predict
> fit202 = arima(ret, order = c(2, 0, 2))
> crpre = predict(fit202, n.ahead = 30)
> dates = seq(as.Date("02/08/2014", format = "%d/%m/%Y"), by =
"days", length = length(ret))
> plot(ret, type = "l", xlim = c(0, 644), ylab = "log return", xlab =
"days",lwd = 1.5)
> lines(crpre$pred, col = "red", lwd = 3)
> lines(crpre$pred + 2 * crpre$se, col = "red", lty = 3, lwd = 3)
> lines(crpre$pred - 2 * crpre$se, col = "red", lty = 3, lwd = 3)
```

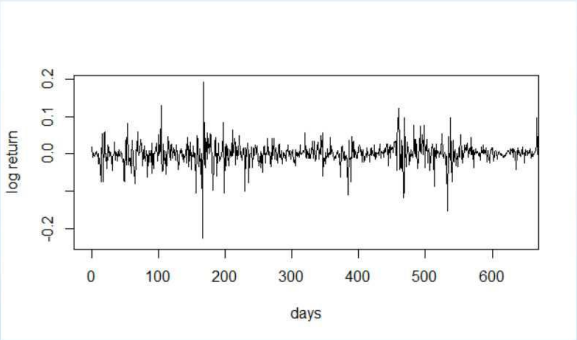


figure7