HW1

1、Below is the increase of memory of PCs over the last 40 years。

Year	Byte
1970	262144
1971	262144
1972	262144
1973	262144
1974	262144
1975	262144
1976	262144
1977	262144
1978	262144
1979	262144
1980	262144
1981	262144
1982	262144
1988	2097152
1989	2097152
1990	1097152
1991	16777216
1992	16777216
1993	16777216
1994	16777216
1995	16777216
1996	268435456
1997	268435456
1998	1073741824
1999	1073741824
2000	1073741824
2004	4294967296
2009	8589934592
2014	17179869184

Logistic regression

- The goal is to model the probability of a random variable Y being 0 or 1 given experimental data.
- Consider a generalized linear model function parameterized by θ , θT=（θ0，θ1， θ2, ... θn）

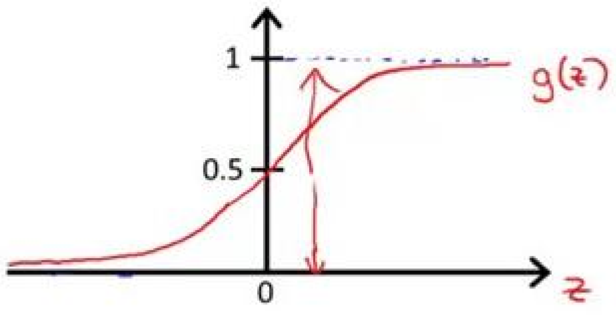
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

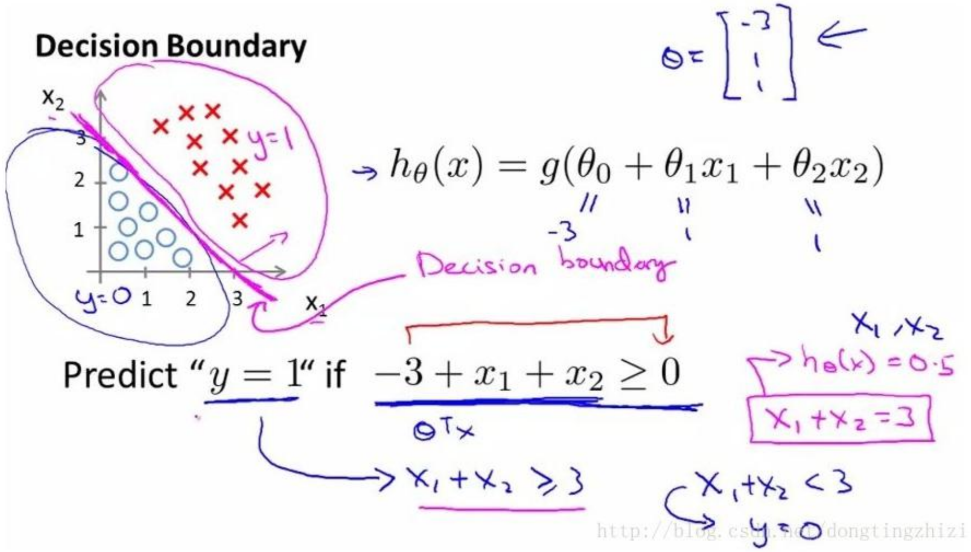
- Outcome

    We predict the *probability* of the outcome occurring;

  θ0， θ1， θ2, ... θn

    Can be thought of in much the same way as multiple regression





The Log-likelihood statistic

- If we attempt to model the probability that y is 0 or 1 with the function

$$h_\theta(X) = P(y=1 \mid X; \Theta) \qquad 1 - h_\theta(X) = P(y=0 \mid X; \Theta)$$

- we take our likelihood function assuming that all the samples are independent,
  - Analogous to the residual sum of squares in multiple regression
  - It is an indicator of how much unexplained information there is after the model has been fitted.
  - Large values indicate poorly fitting statistical models.

$$\begin{aligned} L(\theta|x) &= Pr(Y|X;\theta) \\ &= \prod_i Pr(y_i|x_i;\theta) \\ &= \prod_i h_\theta(x_i)^{y_i}(1-h_\theta(x_i))^{(1-y_i)} \end{aligned}$$

HW2

1、Make an R quantlet to solve HW #1 from unit 1 with R and show it on Github (GH)

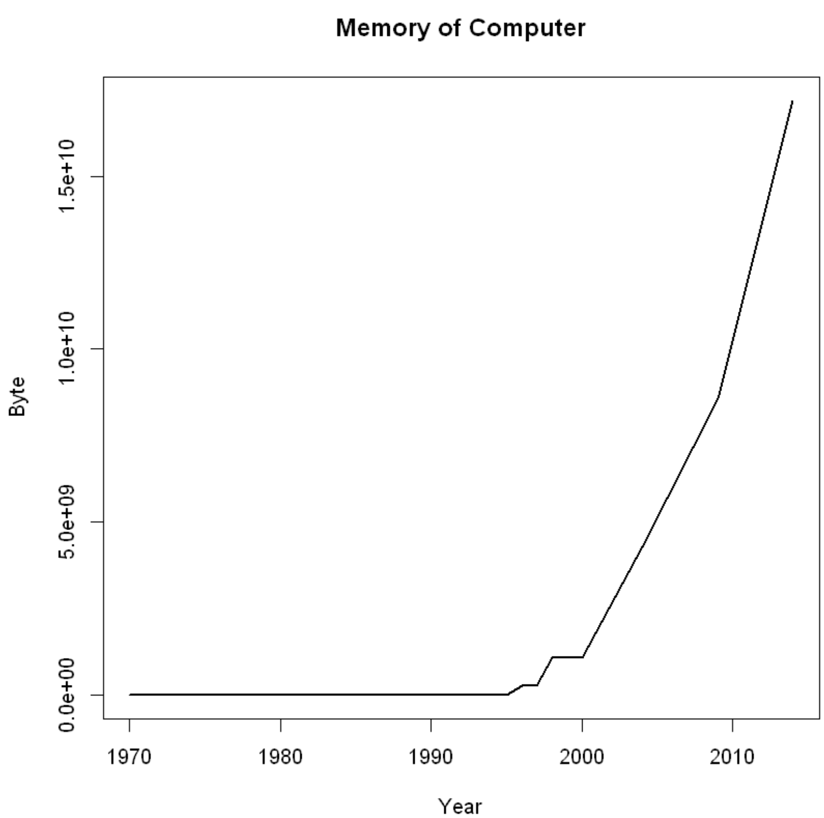
```
In [1]: library(readr)
```

```
Warning message:
"package 'readr' was built under R version 3.3.3"
```

```
In [2]: cpum <- read_csv("cpum.csv",col_names = TRUE)
```

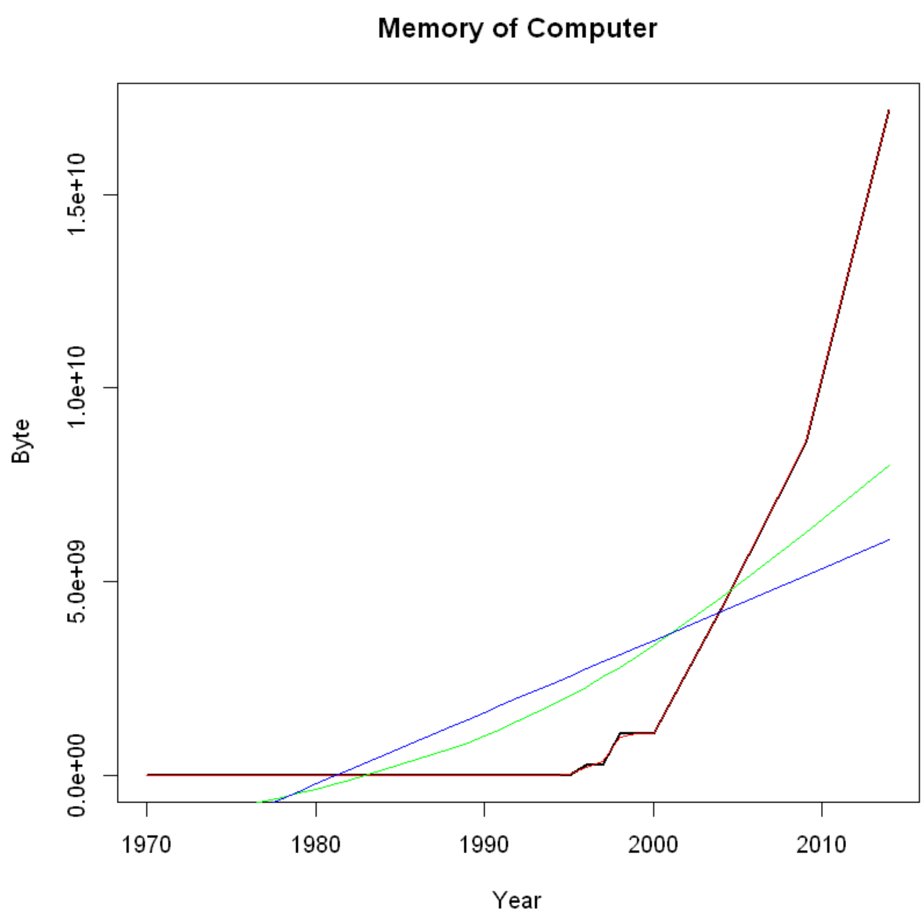
```
Parsed with column specification:
cols(

  Year = col_integer(),
  Byte = col_character()
)
```

```
In [3]: par(mfrow = c(1, 1))
        plot(cpum,type="l",xlab = "Year",ylab = "Byte",lwd=2,main = "Memory of Computer")
```



2、Use R with B-spline code to solve HW#1, any comments?

```
In [4]: plot(cpum,type="l",xlab = "Year",ylab = "Byte",lwd=2,main = "Memory of Computer")

        splines.reg.l1 = smooth.spline(x = cpum$Year, y = cpum$Byte, spar =0.2) # lambda = 0.2
        splines.reg.l2 = smooth.spline(x = cpum$Year, y = cpum$Byte, spar =1) # lambda = 1
        splines.reg.l3 = smooth.spline(x = cpum$Year, y = cpum$Byte, spar =2) # lambda = 2
        lines(splines.reg.l1, col = "red", lwd = 1) # regression line with lambda = 0.2
        lines(splines.reg.l2, col = "green", lwd = 1) # regression line with lambda = 1
        lines(splines.reg.l3, col = "blue", lwd = 1) # regression line with lambda = 2
```



3、**Suppose you observe that in n=1000 mails (in 1 week) you have about 2 scams. Use theLvB /Poisson cdf to calculate that you have 6 scam emails in 2 weeks. In Scammyland you have 5 scams on average, what is the probability to have no scam mail**

```
In [5]: lambda=4
        x=6
        P1=exp(-lambda)*lambda^x/factorial(x)
        P1
```

0.104195634567021

In [6]:
```
lambda=5
x=0
P2=exp(-lambda)*lambda^x/factorial(x)
P2
```

0.00673794699908547

HW 3

1、

```
In [1]: library("digest")
```

```
In [2]: digest("I learn a lot from this class when I am proper listening to the professor", "sha25
```

'c16700de5a5c1961e279135f2be7dcf9c187cb6b21ac8032308c715e1ce9964c'

```
In [3]: digest("I do not learn a lot from this class when I am absent and playing on my Iphone'
```

'2533d529768409d1c09d50451d9125fdbaa6e5fd4efdeb45c04e3c68bcb3a63e'
For the first sentence,the hash number is "c16700de5a5c1961e279135f2be7dcf9c187cb6b21ac8032308c715e1ce9964c"
For the second sentence, the hash number is "2533d529768409d1c09d50451d9125fdbaa6e5fd4efdeb45c04e3c68bcb3a

2、

```
In [4]: library(jsonlite)
```

```
In [5]: json <-

        '[
          {"Name" : "Mario", "Age" : 32, "Occupation" : "Plumber"},
          {"Name" : "Peach", "Age" : 21, "Occupation" : "Princess"},
          {},
          {"Name" : "Bowser", "Occupation" : "Koopa"}
        ]'
```

```
In [6]: data_frame <- fromJSON(json)
        data_frame
```

Name	Age	Occupation
Mario	32	Plumber
Peach		Princess
~~NA~~	~~NA~~	~~NA~~
Bowser	NA	Koopa

21

```
In [7]: data_frame$Ranking <- c(3, 1, 2, 4)
        data_frame
```

Name	Age	Occupation	Ranking
Mario	32	Plumber	3
Peach	21	Princess	1
NA	NA	NA	2
Bowser	NA	Koopa	4

```
In [8]: toJSON(data_frame, pretty=TRUE)
```

```
[
  {
    "Name": "Mario",
    "Age": 32,
    "Occupation": "Plumber",
    "Ranking": 3
  },
  {
    "Name": "Peach",
    "Age": 21,
    "Occupation": "Princess",
    "Ranking": 1
  },
  {
    "Ranking": 2
  },
  {
    "Name": "Bowser",
    "Occupation": "Koopa",
    "Ranking": 4
  }
]
```

```
In [9]: write_json(json,path="C:/Users/Aiqing-Jiang/1.json")

In [10]: read_json(path="C:/Users/Aiqing-Jiang/1.json",simplifyVector = FALSE)
```

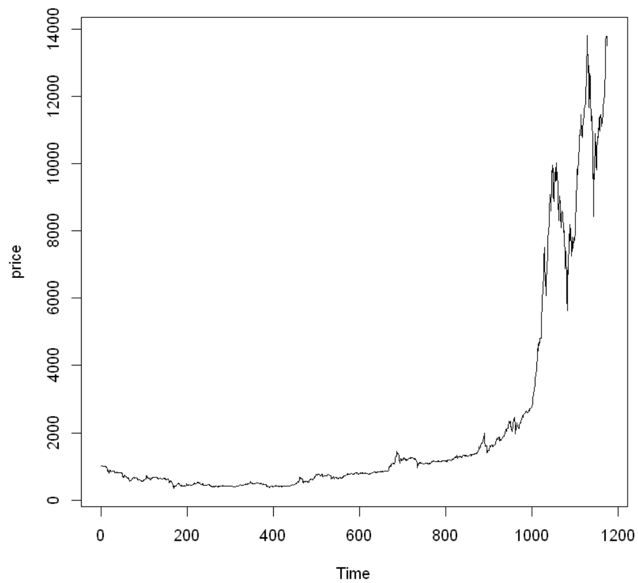
1. '[{"Name" : "Mario", "Age" : 32, "Occupation" : "Plumber"}, {"Name" : "Peach", "Age" : 21, "Occupation" : "Princess"}, {}, {"Name" : "Bowser", "Occupation" : "Koopa"}]'

3、

```
In [12]: library(rjson)

In [13]: json_file = "http://crix.hu-berlin.de/data/crix.json"
         json_data = fromJSON(file=json_file)

In [14]: crix_data_frame = as.data.frame(json_data)

In [15]: a <- 1:1175
         n <- 2*a
         m <- n-1
```

```
In [16]: date <- t(crix_data_frame[m])
         price <- t(crix_data_frame[n])
         crix_data <- cbind(date,price)
```

```
In [17]: ts.plot(price)
```

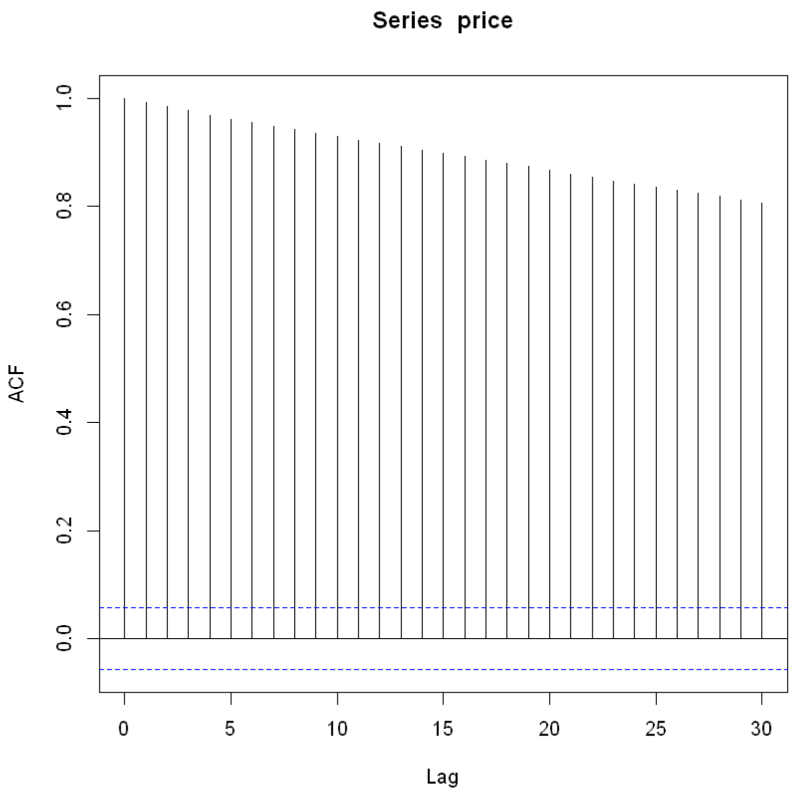



```
In [20]: library(tseries)
```

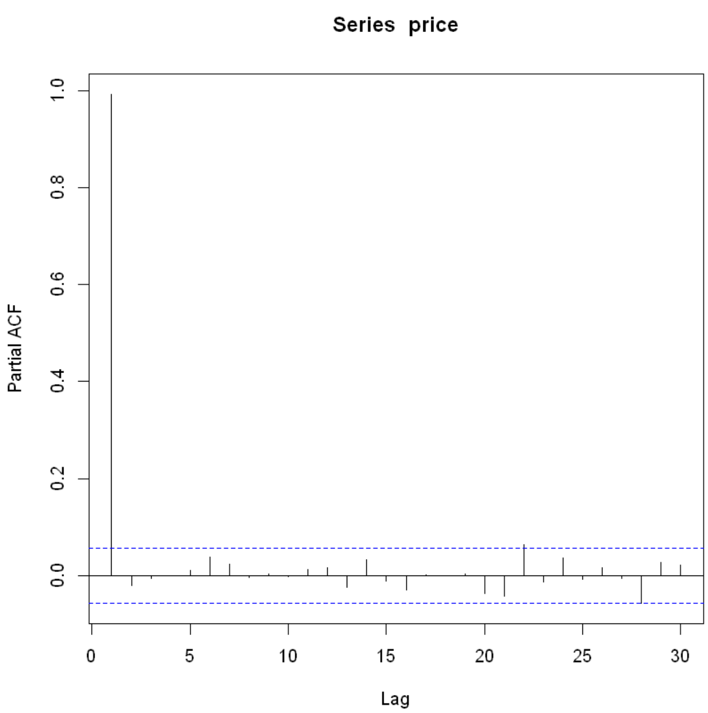
```
In [21]: adf.test(price)
```

```
Warning message in adf.test(price):
"p-value greater than printed p-value"

Augmented Dickey-Fuller Test

data:  price
Dickey-Fuller = 0.47023, Lag order = 10, p-value = 0.99
alternative hypothesis: stationary
```

```
In [22]: acf(price)
```




```
In [23]: pacf(price)
```



```
In [25]: library(forecast)
```

```
In [26]: auto.arima(price)# ARIMA(5,2,0)
```

```
Series: price
ARIMA(5,2,0)

Coefficients:
          ar1      ar2      ar3      ar4      ar5
      -0.8808  -0.7101  -0.5786  -0.4783  -0.2543
s.e.   0.0284   0.0359   0.0380   0.0362   0.0286

sigma^2 estimated as 32821:  log likelihood=-7761.48

AIC=15534.95   AICc=15535.02   BIC=15565.36
```

Digital Signature Algorithm

The Digital Signature Algorithm (DSA) can be used by the recipient of a message to verify that the message has not been altered during transit as well as ascertain the originator’s identity. A digital signature is an electronic version of a written signature in that the digital signature can be used in proving to the recipient or a third party that the message was, in fact, signed by the originator. Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at any later time.

Digital Signature Generation and Verification

- The DSA is used by a signatory to generate a digital signature on data and by a verifier to verify the authenticity of the signature. Each signatory has a public and private key. The private key is used in the signature generation process and the public key is used in the signature verification process. For both signature generation and verification, the data (which is referred to as a message) is reduced by means of the Secure Hash Algorithm (SHA) specified in FIPS 180-1. An adversary, who does not know the private key of the signatory, cannot generate the correct signature of the signatory. In other words, signatures cannot be forged. However, by using the signatory’s public key, anyone can verify a correctly signed message.

DSA Standard

- The Digital Signature Algorithm is a United States Federal Government standard for digital signatures. DSA was 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. A minor revision was issued as FIPS 186-1, and the standard was expanded further as FIPS 186-2. DSA , as described in U.S. Patent 5,231,668, is attributed to David W. Kravitz, a former National Security Agency (NSA) employee. The NIST has made this patent available world-wide royalty-free. The standard continues to be revised and updated periodically by NIST.

HW 4

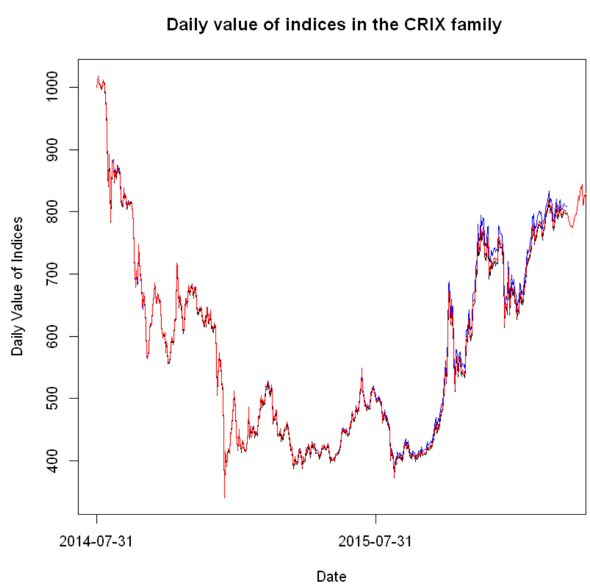
```
In [1]: library('rjson')

In [2]: json_file = "http://crix.hu-berlin.de/data/crix.json"

       json_data = fromJSON(file=json_file)
       crix_data_frame=as.data.frame(json_data)

In [3]: x=crix_data_frame
        dim(x)
```

1. 1 2. 2356

```
In [4]: n=dim(x)
        a=seq(1,n[2],2)
        b=seq(2,n[2],2)

In [5]: date=t(x[1,a])
        price=t(x[1,b])

In [6]: crix=data.frame(date,price)

In [7]: load("ecrix.RData")
        load("efcrix.RData")
```

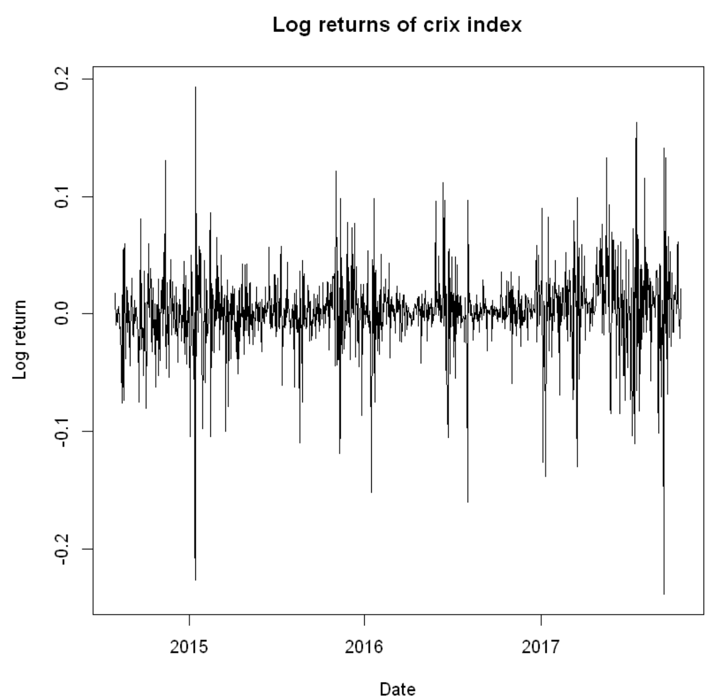
1、 Daily value of indices in the CRIX family

```
In [8]: plot(ecrix, type = "l", col = "blue", xaxt = "n",main = " Daily value of indices in th
        lines(efcrix, col = "black")
        lines(price, col = "red")
        lab=seq(1,n[2],365)
        axis(1, at = lab, label = names(ecrix)[lab])
```



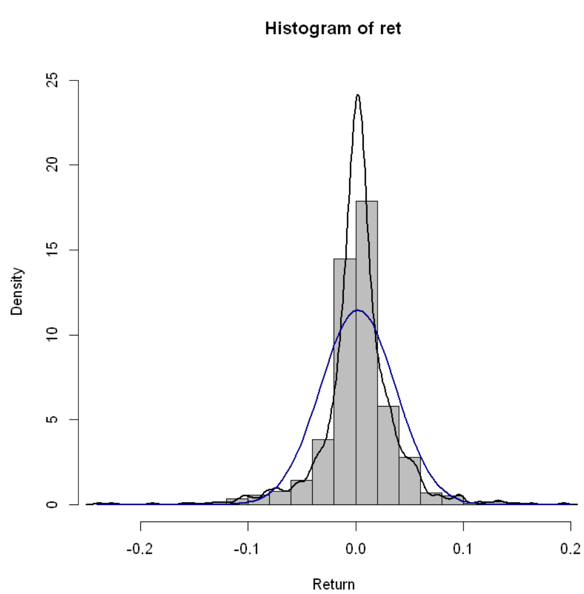
2、 The log returns of CRIX index In [9]:

```
ret=diff(log(price))
    plot(ret~as.Date(date[-1]), type="l", col="black", xlab="Date", ylab="Log return", main="Log
```

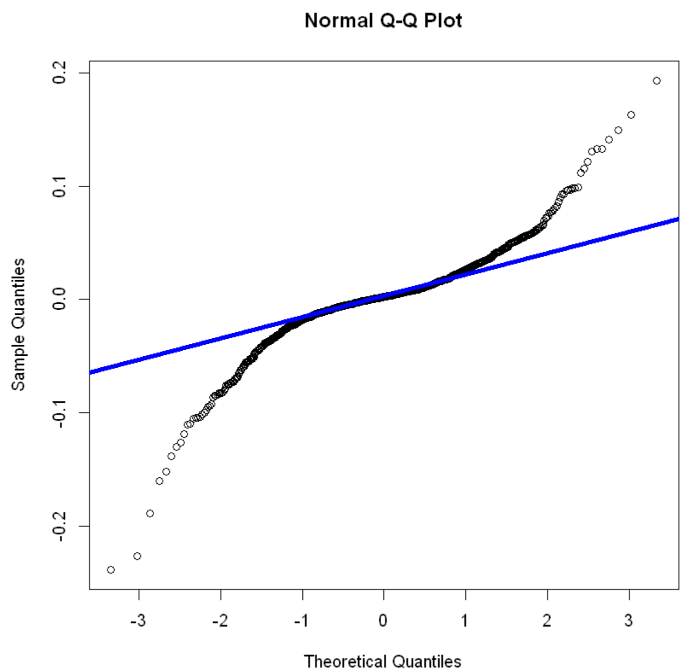


3、 Histogram and QQ plot of CRIX returns

```
In [10]: hist(ret, col = "grey", breaks = 20, freq = FALSE, ylim = c(0, 25), xlab = "Return")

     lines(density(ret), lwd = 2)
     x = seq(-4, 4, length = 100)
     curve(dnorm(x, mean = mean(ret), sd = sd(ret)), add=TRUE, col = "darkblue", lwd = 2)
```

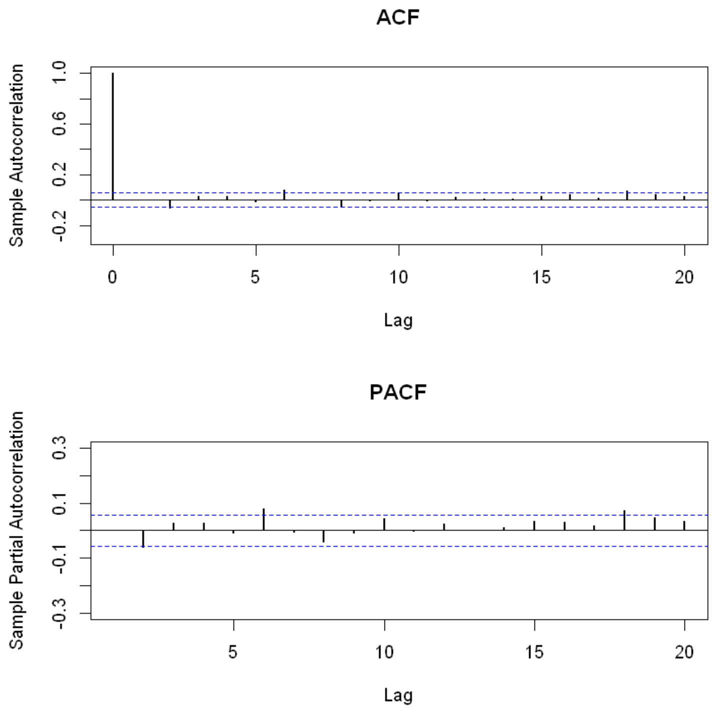


```
In [11]: qqnorm(ret)
         qqline(ret, col = "blue", lwd = 4)
```



4、 The sample ACF and PACF of CRIX returns

```
In [13]:
par(mfrow = c(2, 1))


libraries = c("zoo", "tseries")
autocorr = acf(ret, lag.max = 20, ylab = "Sample Autocorrelation",
               main = "ACF" ,
               lwd = 2, ylim = c(-0.3, 1))
autopcorr = pacf(ret, lag.max = 20, ylab = "Sample Partial Autocorrelation",
                 main = "PACF" ,
                 ylim = c(-0.3, 0.3), lwd = 2)
```



5、 Figure 7:Diagnostic Checking

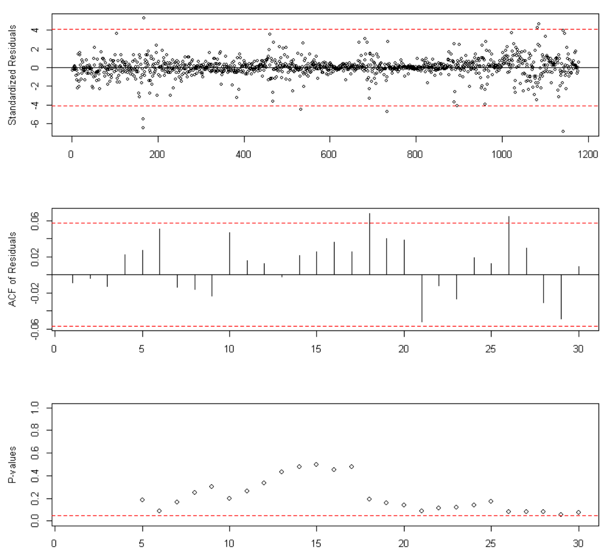
In [15]:
```
library(TTR)
library(TSA)
library(caschrono)
library(forecast)
```

In [16]:
```
auto.arima(ret)
```

```
Series: ret
ARIMA(1,1,0) with drift

Coefficients:
```

```
         ar1   drift
     -0.4695  0e+00
s.e.  0.0257  9e-04

sigma^2 estimated as 0.001881:  log likelihood=2022.35
AIC=-4038.7   AICc=-4038.68   BIC=-4023.49
```

```
In [17]: fit = arima(ret, order = c(2, 0, 2))
         tsdiag(fit)
```




```
In [18]: par(mfrow = c(2, 1))
         crix_pre = predict(fit, n.ahead = 30)

#dates = seq(as.Date("31/07/2014", format = "%d/%m/%Y"), by = "days", length = length( ret))
plot(ret, type = "l", ylab = "Log return", xlab = "Date",
    lwd = 1, main = "CRIX returns and predicted values")
lines(crix_pre$pred, col = "red", lwd = 1)
lines(crix_pre$pred + 2 * crix_pre$se, col = "red", lty = 3, lwd = 1)
lines(crix_pre$pred - 2 * crix_pre$se, col = "red", lty = 3, lwd = 1)
```