Statistics, Time Series, Computation Finance, Derivative Pricing, Algorithmic Trading Review in R, Python

Ron Wu

Last update: 4/25/16

Table of Contents

Statistics

1. Distributions

- Random Uniform

linear congruential generator

$$X_{n+1} = (aX_n + c) \bmod m$$

Park-Miller pseudo-random number generator is a typical choice of a, c, m

See Mark Joshi ParkMiller.cpp

How C/C++ generates $U(0,1)$?

GNU C Library source:
https://sourceware.org/git/?p=glibc.git;a=blob_plain;f=stdlib/rand_r.c;hb=HEADe

```
int rand_r(unsigned int *seed)
{
	unsigned int next = *seed;
	int result;

	next *= 1103515245;
	next += 12345;
	result = (unsigned int)(next / 65536) % 2048;

	next *= 1103515245;
	next += 12345;
	result <<= 10;
	result ^= (unsigned int)(next / 65536) % 1024;

	next *= 1103515245;
	next += 12345;
```

```
        result <<= 10;
        result ^= (unsigned int)(next / 65536) % 1024;

        *seed = next;

        return result;
}
```

How R generates $U(a, b)$?

-R

```
runif(10)  #between a=0, b=1  with var = (b-a)^2/12 'r' stands for random
```

R source code wrap in C: https://github.com/wch/r-source/blob/e5b21d0397c607883ff25cca379687b86933d730/src/nmath/standalone/sunif.c

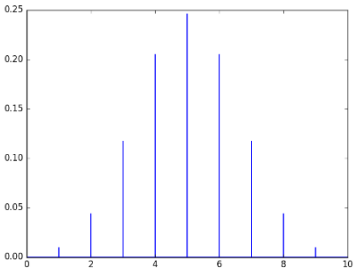
```
double runif(double a, double b)
{

        double u;
        do { u = unif_rand(); } while (u <= 0 || u >= 1);
        return a + (b - a) * u;
}


/* A version of Marsaglia-MultiCarry */

static unsigned int I1 = 1234, I2 = 5678;

void set_seed(unsigned int i1, unsigned int i2)
{
        I1 = i1; I2 = i2;
}

void get_seed(unsigned int *i1, unsigned int *i2)
{
        *i1 = I1; *i2 = I2;
}


double unif_rand(void)
{
        I1 = 36969 * (I1 & 0177777) + (I1 >> 16); //I1*2^-16
        I2 = 18000 * (I2 & 0177777) + (I2 >> 16);
        return ((I1 << 16) ^ (I2 & 0177777)) * 2.328306437080797e-10; //XOR
        /* [0,1) */
}
```

- Binomial

```
#prob of flipping n=10 coins and getting exact k=5 heads, given p=0.5
```

```
pbinom(5,size = 10, prob = .5, lower.tail = TRUE)-pbinom(4,size = 10,
prob = .5, lower.tail = TRUE)

#' mean = np
#' var = np(1-p)
```

```
import numpy as np
from scipy.stats import binom
import matplotlib.pyplot as plt

n, p = 10, 0.5
x = np.arange(0, 10, 0.001)
plt.plot(x, binom.pmf(x, n,p))
plt.show()
```



- Normal

Generate Normal—Box-Muller

$$z_0 = \sqrt{-2\ln U_1}\cos(2\pi U_2) = \sqrt{-2\ln s}\left(\frac{u}{\sqrt{s}}\right) = u\sqrt{\frac{-2\ln s}{s}}$$

Where $U_{1,2}$ are $U(0,1)$, and $u, v = 2U_{1,2} - 1, s = u^2 + v^2$

See Mark Joshi Random1.cpp

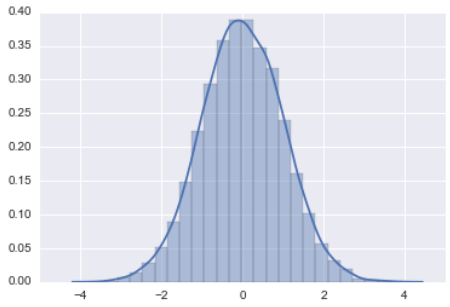
Use polynomial approximation to c.d.f. and its inverse (Hastings) to get random normal number

See Mark Joshi Normal.cpp

Or use C++ gsl library for handling Gaussian stuff
https://www.gnu.org/software/gsl/manual/html_node/The-Gaussian-Distribution.html
It uses Ziggurat Algorithm.

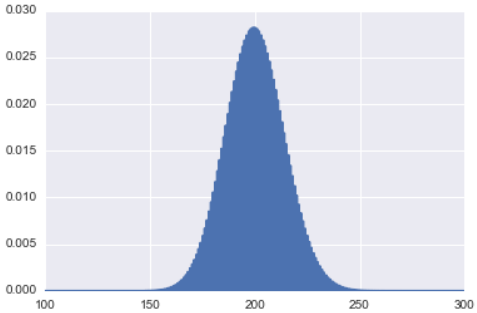
```
x <- rnorm (100)

#' mean = 0
#' var = 1

summary(x)

pnorm(1.69) #Pr=P(x<1.69) = 95.4486%, 'p' stands for p-value
```

```
import numpy as np
import seaborn as sns

vals = np.random.randn(10000)
sns.distplot(vals,bins=25)
```



- Poisson
  Let binomial let n --> infinity, application in contingency table

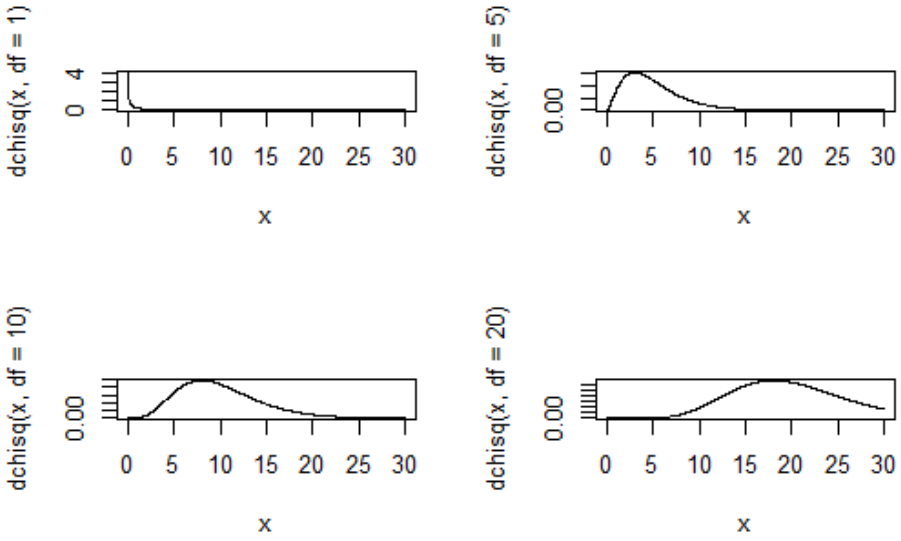
```
#mean = var = Lambda

#say during one hour window an event happens
#200 times on average, what is the prob of that
#event happens for exactly 190 times
ppois(190, lambda = 200,lower.tail = TRUE)-ppois(189, lambda =
200,lower.tail = TRUE)

import numpy as np
from scipy.stats import poisson
import matplotlib.pyplot as plt

mu = 200
x = np.arange(100, 300, 0.5)
plt.plot(x, poisson.pmf(x, mu))
```



- Chi-square
  $Z_{1,2,3}$, ... in iid $N(0,1)$, then $Z_1^2 + \ldots + Z_k^2 \sim \chi^2(k), k$ = d.f.

$E(Z_1^2 + ... + Z_k^2) = k, \chi^2(k)$ → normal as k → inf

```
x <- seq(0, 30, 0.01)
par(mfrow=c(2,2))
plot(x, dchisq(x, df=1), type="l")
plot(x, dchisq(x, df=5), type="l")
plot(x, dchisq(x, df=10), type="l")
plot(x, dchisq(x, df=20), type="l")
```



- Student's T

$Z \sim N(0,1), X \sim \chi^2(k)$, then

$T := \frac{Z}{\sqrt{\left(\frac{X}{k}\right)}}$ is T-distribution with d.f. = k

$$E(T) = 0$$

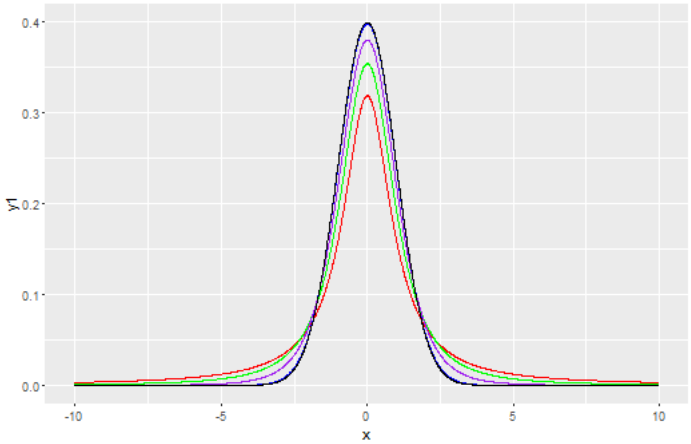
$$Shew(T) = 0$$

$$Kurt(T) = (3k-6)/(k-4), for\ k > 4$$

T → N(0,1) as k → inf, effective-inf when k>60

```
x <- seq(-10, 10, 0.01)
y1 <- dt(x, df=1)
y2 <- dt(x, df=2)
y3 <- dt(x, df=5)
y4 <- dt(x, df=60)
y.norm <- dnorm(x)

df <- data.frame(x,y1,y2,y3,y4,y.norm)

library(ggplot2)
ggplot(df, aes(x)) +
```

```
      geom_line(aes(y=y1), colour="red") +
      geom_line(aes(y=y2), colour="green") +
      geom_line(aes(y=y3), colour="purple") +
      geom_line(aes(y=y4), colour="blue") +
      geom_line(aes(y=y.norm), colour="black")
```



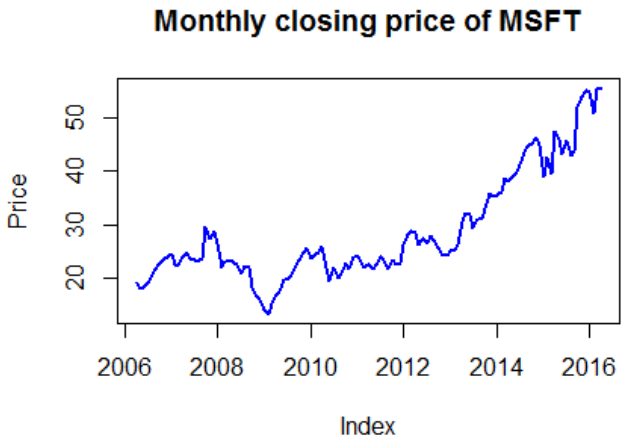
- F

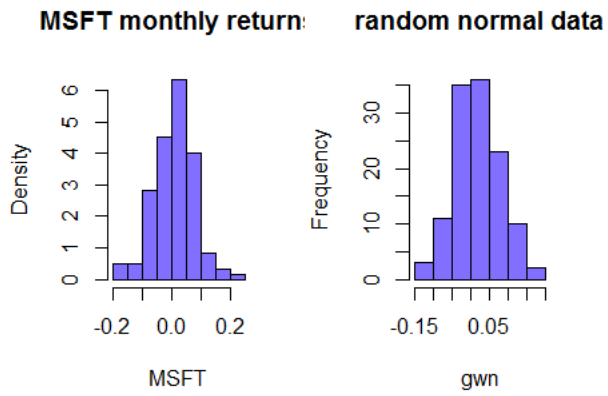
$$X \sim \chi^2(m),\ Y \sim \chi^2(n), \text{ then}$$
$$(X/m)\ /\ (Y/n) \sim F(m,n)$$

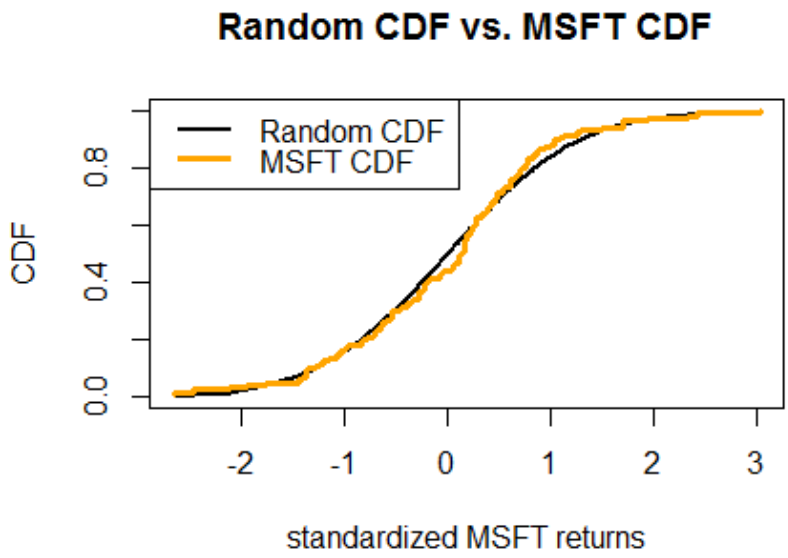
- Show stock return close to normal rv

```
library(PerformanceAnalytics)
library(tseries)
library(zoo)

end = Sys.Date()
d<- as.POSIXlt(as.Date(end))
d$year <- d$year - 10
start = as.Date(d)
par(mfrow=c(1,1))
#monthly adj closed price 10 year
MSFT.prices = get.hist.quote(instrument="msft", start=start,
                             end=end, quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")

plot(MSFT.prices,main="Monthly closing price of MSFT",
     ylab="Price", lwd=2, col="blue")
MSFT = diff(log(MSFT.prices))
plot(MSFT, plot.type="single", col="red",
     main="Monthly cc returns on MSFT ",  lwd=2)
abline(h=0)
#we will show log return not the usual return close to random normal

set.seed(1)
gwn = rnorm(length(MSFT),mean=mean(MSFT),sd=sd(MSFT))
par(mfrow=c(1,2))
plot(MSFT,main="Monthly cc returns on MSFT", lwd=2, col="blue")
```

```
abline(h=0)
ts.plot(gwn,main="Random Normal with the same mean, sd", lwd=2, col="blue")
abline(h=0)


# histogram compare
par(mfrow=c(1,2))
hist(MSFT,main = "MSFT monthly returns",
     probability=TRUE, col="slateblue1")
hist(gwn, main="random normal data", col="slateblue1")

par(mfrow=c(1,1))
# compare empirical cdf to standard normal cdf for MSFT returns
MSFT.mat = coredata(MSFT)
colnames(MSFT.mat) = "MSFT"
rownames(MSFT.mat) = as.character(index(MSFT))
z1 = scale(MSFT.mat)
n1 = length(MSFT.mat)
F.hat = 1:n1/n1
x1 = sort(z1)
y1 = pnorm(x1)

plot(x1,y1,main="Random CDF vs. MSFT CDF",
     type="l",lwd=2,xlab="standardized MSFT returns",ylab="CDF")
points(x1,F.hat, type="s", lty=1, lwd=3, col="orange")
legend(x="topleft",legend=c("Random CDF","MSFT CDF"),
       lty=c(1,1), lwd=c(2,3), col=c("black","orange"))
```







2. Moments, Mode, Covariance, Correlation

- Moments

  mean, variance, skewness, $\sum[(x-\bar{x})^3/\sigma^3]/n-1$, kurtosis $\sum[(x-\bar{x})^4/\sigma^4]/n-1$

```
import numpy as np
import scipy.stats as sp
import matplotlib.pyplot as plt

vals = np.random.randint(0, 100, 10000)
plt.hist(vals, 50)
plt.show()

np.mean(vals)
np.var(vals)
sp.skew(vals)
sp.kurtosis(vals)

from scipy import stats
stats.mode(vals)
```

- Mode

```
from scipy import stats
stats.mode(vals)
```

- Constant expected return model

Retrieve 4 stocks : Open, High, Low, Close, Volume, Adj Close

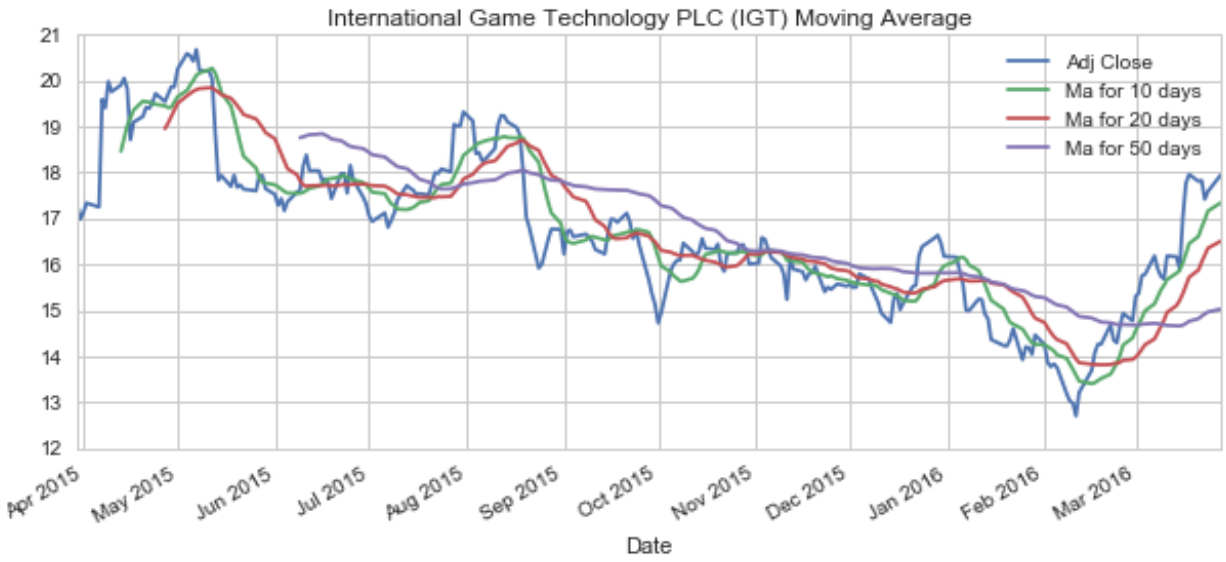
The following code taken from https://nbviewer.ipython.org/github/jmportilla/Udemy-notes/blob/master/Data%20Project%20-%20Stock%20Market%20Analysis.ipynb

```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from datetime import datetime
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')

#Barron's Stock Picks for 3/28/2016
barron_list = ['IGT','IPG','ZBH','RELY']

end = datetime.now()
start = datetime(end.year - 1 , end.month, end.day)

for stock in barron_list:
    globals()[stock]=web.DataReader(stock, 'yahoo',start, end)

IGT['Adj Close'].plot(legend=True, figsize=(10,4),
    title="International Game Technology PLC(IGT) Adjusted Closing Price ")
```

- Plot moving average

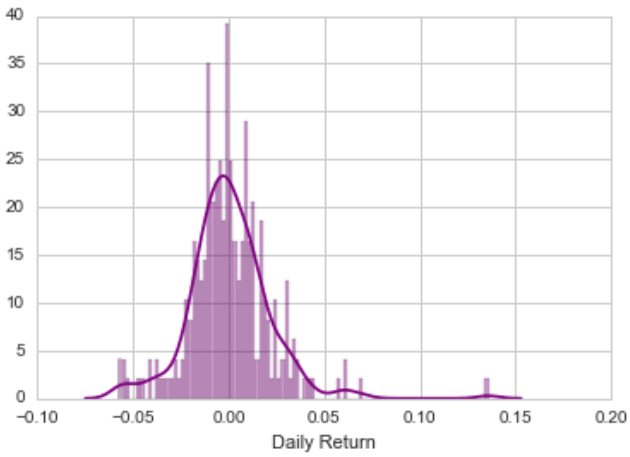
```
ma_day = [10,20,50] #simple moving average

for ma in ma_day:
    column_name = 'Ma for %d days' %((ma))
    IGT[column_name] = pd.rolling_mean(IGT['Adj Close'], ma)

IGT[['Adj Close', 'Ma for 10 days',
```

```
        'Ma for 20 days', 'Ma for 50 days']].plot(subplots = False
              , title="International Game Technology PLC (IGT) Moving Average")
```



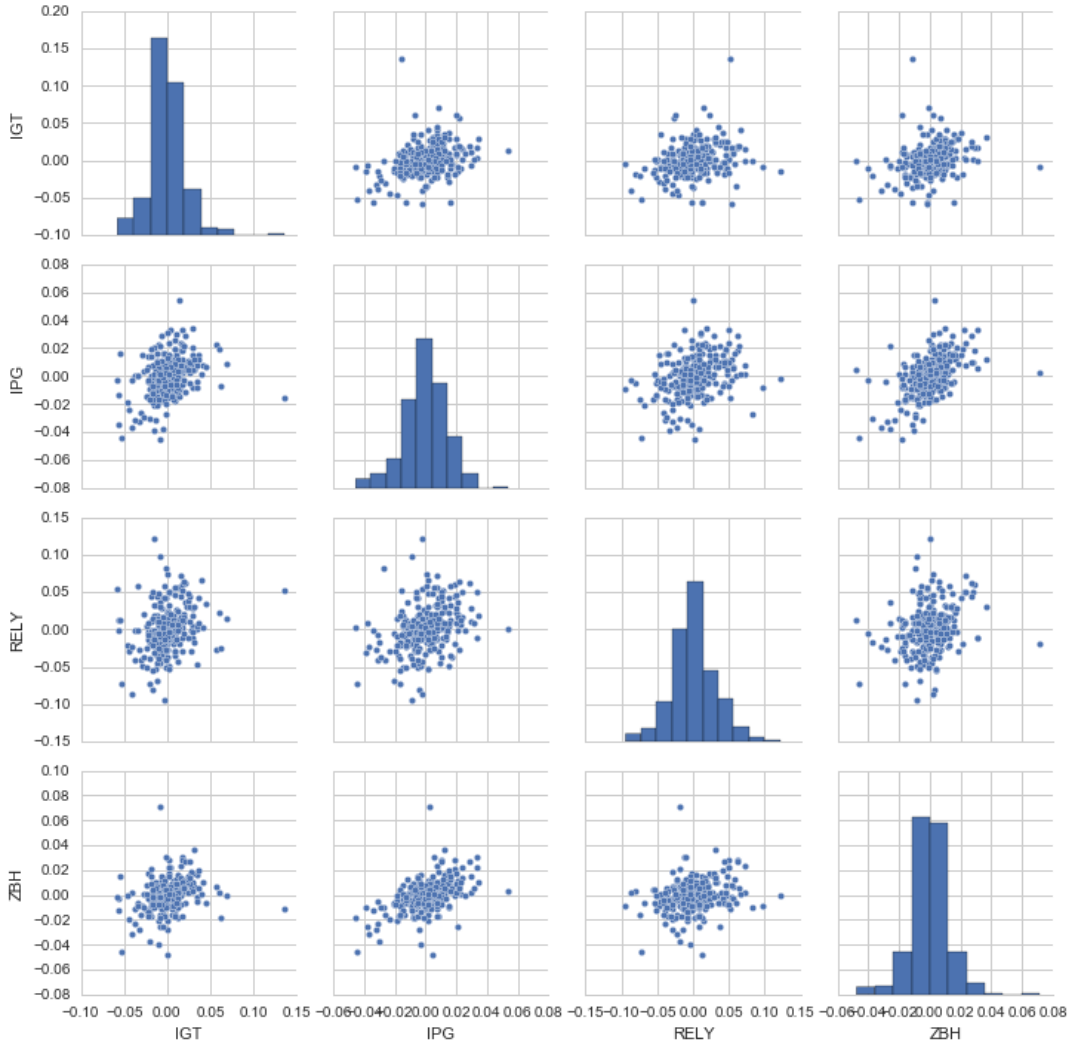
- Return histogram

```
IGT['Daily Return'] = IGT['Adj Close'].pct_change()
sns.distplot(IGT['Daily Return'].dropna(), bins =100, color = 'purple')
```



- Correlation of the 4 stocks returns

```
closeing_df = web.DataReader(barron_list, 'yahoo', start, end)['Adj Close']
barron_rets = closeing_df.pct_change()
sns.pairplot(barron_rets.dropna())
```

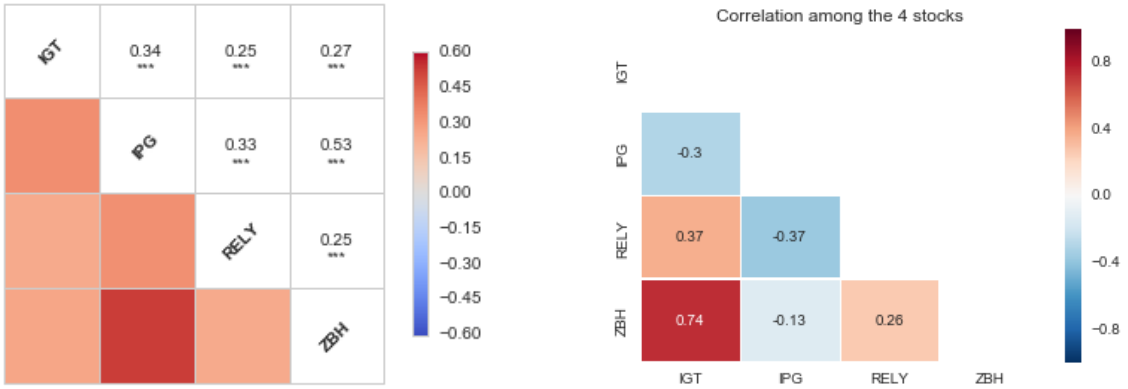


- Correlation plot

```
sns.corrplot(barron_rets.dropna(), annot=True)
```

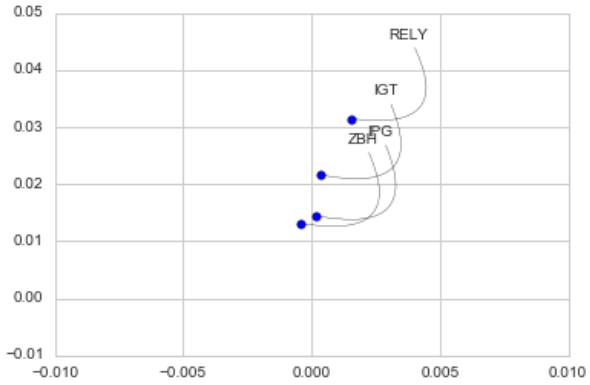
or

```
corr = closeing_df.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(corr.dropna(),mask = mask,linewidths=.2, annot=True)
plt.title('Correlation among the 4 stocks')
```



- Risk return comparison among the portfolio

```
rets=barron_rets.dropna()
area = np.pi*10
plt.scatter(rets.mean(), rets.std(), s= area)
for label, x, y in zip(rets.columns, rets.mean(), rets.std()):
    plt.annotate(
        label,
        xy = (x, y), xytext = (50, 50),
        textcoords = 'offset points', ha = 'right', va = 'bottom',
        arrowprops = dict(arrowstyle = '-', connectionstyle = 'arc3,rad=-1.0'))
```



- Value at Risk

  Bootstrap way—gives 3.33%

```
barron_rets['IGT'].quantile(0.05)
#-0.033358568359280846
#showing based on past one year performance, there is 5% chance
#of losing 3.33% or more on a single day.
```

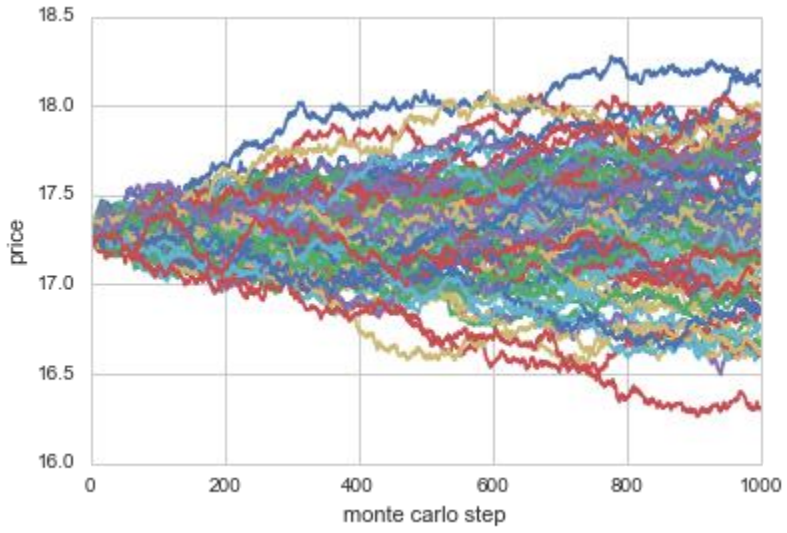
- Value at Risk – Monte Carlo way (using geometric random walk) —gives 3.47%

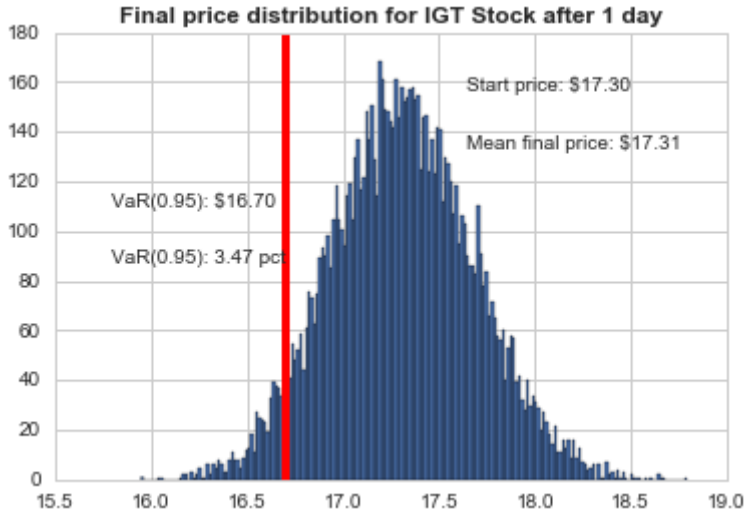
```
steps = 1000
dt = 1/steps
mu = barron_rets.mean()['IGT']
sigma = barron_rets.std()['IGT']

def stock_monte_carlo(start, steps, mu, sigma):
    price = np.zeros(steps)
```

```
    price[0] = start_price

    stock = np.zeros(steps)
    drift = np.zeros(steps)

    for x in range(1, steps):
        stock[x] = np.random.normal(loc = mu*dt, scale = sigma * np.sqrt(dt))
        drift[x] = mu * dt
        price[x] = price[x-1] + (price[x-1] * (drift[x]+stock[x]))

    return price

start_price = closeing_df['IGT'][0]
for run in range(100):
    plt.plot(stock_monte_carlo(start_price, steps, mu, sigma))

plt.ylabel('price')
plt.xlabel('monte carlo step')
plt.show()

runs = 10000
simulation = np.zeros(runs)

for run in range(runs):
    simulation[run] = stock_monte_carlo(start_price, steps, mu, sigma)[steps-1]
q = np.percentile(simulation, 5)
plt.hist(simulation, bins= 200)
plt.figtext(0.6, 0.8, s="Start price: $%.2f" %start_price)
plt.figtext(0.6, 0.7, "Mean final price: $%.2f" % simulation.mean())

# Display 5% quantile
plt.figtext(0.19, 0.6, "VaR(0.95): $%.2f" % q)
qq = (start_price-q)/start_price * 100
plt.figtext(0.19, 0.5, "VaR(0.95): %.2f pct" % qq)
plt.axvline(x=q, linewidth=4, color='r')
plt.title(u"Final price distribution for IGT Stock after 1 day", weight='bold');
```

In the code we put "`scale = sigma * np.sqrt(dt))`" in the grw is to match this $se(\hat{\mu}_i) = SD(\hat{\mu}_i) = \sigma_i/\sqrt{T}$ where T = time step in the example

$se(value\ at\ risk) = variance(exp(quantile\_estimate) - 1)$ see standard error of estimate and boostrapping error





- Multi-Asset Constant Expected Return Simulation

Assume: normal and covariance stationary ($\mu_i, \sigma_i, \sigma_{ij}$ are all constant over time, white noise are uncorrelated over time)

```
library(PerformanceAnalytics)
library(tseries)
library(zoo)

end = Sys.Date()
d<- as.POSIXlt(as.Date(end))
d$year <- d$year - 10
start = as.Date(d)
par(mfrow=c(1,1))
#monthly adj closed price 10 year
MSFT.prices = get.hist.quote(instrument="MSFT", start=start,
                             end=end, quote="AdjClose",
```

```
                              provider="yahoo", origin="1970-01-01",
                              compression="m", retclass="zoo")

AMZN.prices = get.hist.quote(instrument="AMZN", start=start,
                              end=end, quote="AdjClose",
                              provider="yahoo", origin="1970-01-01",
                              compression="m", retclass="zoo")


GOOG.prices = get.hist.quote(instrument="GOOG", start=start,
                              end=end, quote="AdjClose",
                              provider="yahoo", origin="1970-01-01",
                              compression="m", retclass="zoo")

MSFT = diff(log(MSFT.prices))
AMZN = diff(log(AMZN.prices))
GOOG = diff(log(GOOG.prices))

totReturn = merge(merge(MSFT, AMZN),GOOG)
index(totReturn)=as.yearmon(index(totReturn))
colnames(totReturn) = c("MSFT","AMZN","GOOG")

pairs(totReturn, col="blue")

mu = apply(totReturn, 2, mean)
sigma_ij2 = cov(totReturn)
#           MSFT     AMZN     GOOG
# MSFT 0.004990 0.002814 0.003070
# AMZN 0.002814 0.011459 0.004610
# GOOG 0.003070 0.004610 0.007151

#simulate stocks returns 100 times
library(mvtnorm)
nobs = 100
set.seed(123)
returns.sim = rmvnorm(nobs, mean=mu, sigma=sigma_ij2)
colnames(returns.sim) = c("MSFT", "AMZN", "GOOG")

cov(returns.sim)
#           MSFT     AMZN     GOOG
# MSFT 0.003678 0.002384 0.002232
# AMZN 0.002384 0.010101 0.003727
# GOOG 0.002232 0.003727 0.006764


weight <- matrix(c(.3,.3,.4), ncol = 1)

port.var = t(weight)%*%sigma_ij2%*%weight; port.var
#0.004974368 the var is smaller than any of the stock alone
diag(sigma_ij2)
#       MSFT        AMZN        GOOG
#0.004990020 0.011458789 0.007151222
```

To generate correlated normal random variables above, r has rmvnorm(n, sigma), which uses one of the 3 methods:

Eigen:

```
ev <- eigen(sigma, symmetric = TRUE)
t(ev$vectors %*% (t(ev$vectors) * sqrt(ev$values))) %*% matrix(rnorm(n *
ncol(sigma)), n)
```

Svd:

```
s <- svd(sigma)
t(s$v %*% (t(s$u) * sqrt(s$d))) %*% matrix(rnorm(n * ncol(sigma)), n)
```

Cholesky decomposition:

$$\sigma = t(U)D(U) = t(R)(R),$$

where U = upper triangular, D = diagonal, R = sqrt(D)U

```
R <- chol(sigma, pivot = TRUE)  #use pivot, b/c sigma is positive-definite
t(R[, order(attr(R, "pivot"))] ) %*% matrix(rnorm(n * ncol(sigma)), n)
```

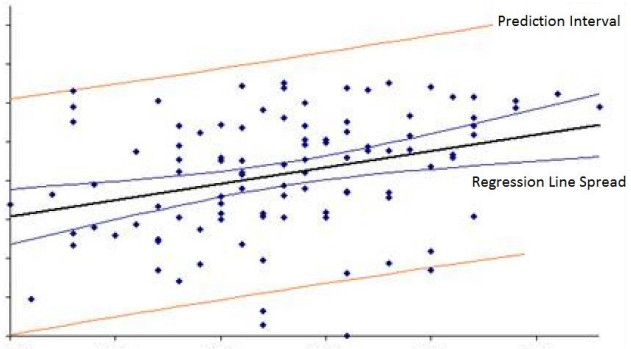
3. Regression

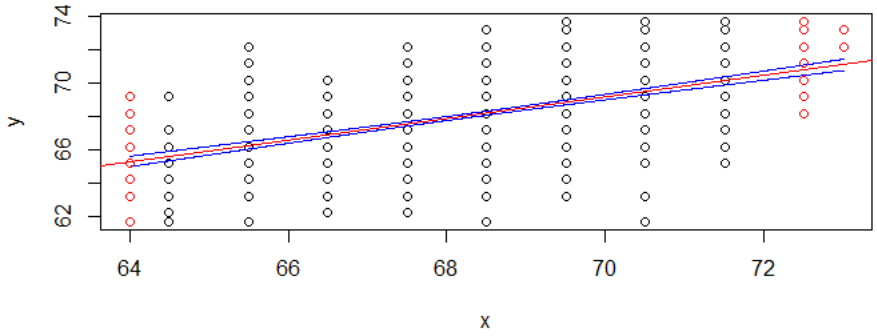
- 1D linear regression

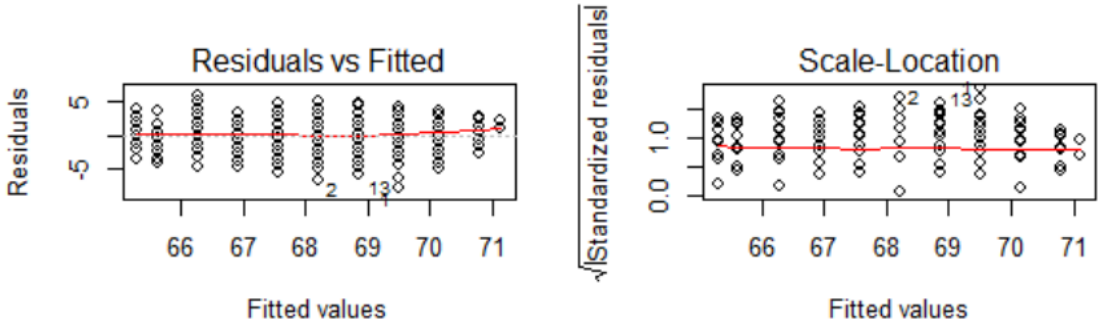
```
library(UsingR); data(galton) #predict son's height from father's
y<-galton$child
x<-galton$parent
length(x) #928

fit<-lm(y~x)
coef(fit)
#which is given by
beta <- cor(y,x)* sd(y) / sd(x)
alpha <- mean(y) - beta * mean(x)

#residual
e_hat<-resid(fit)
y_hat<-predict(fit)
ssx <- sum((x-mean(x))^2)

#standard error of estimate
sse<-sqrt(sum(e_hat^2)/(length(e_hat)-2))
#2.238547
#or
summary(fit)$sigma

#Standard Error of slope: sqrt(var(beta))
se_beta <- sse / sqrt(ssx)
#t_statistics of slope
t_beta <- beta / se_beta
```

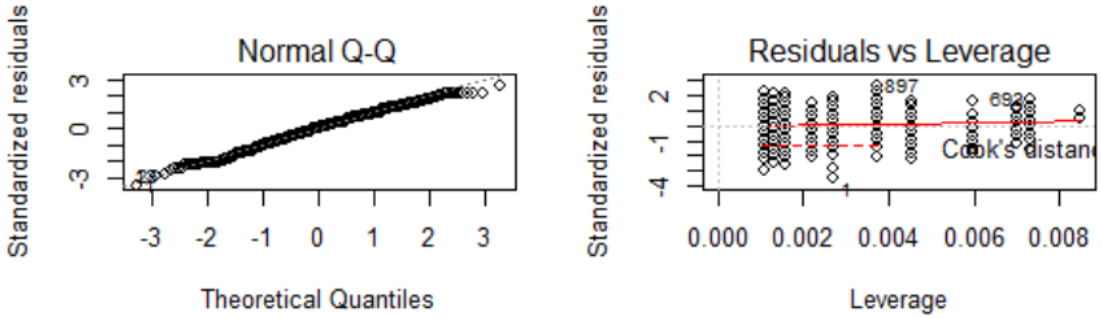
```r
p_beta <- 2 * pt(abs(t_beta),df = n-2, lower.tail = F)

#Standard Error of intersect: sqrt(var(alpha))
se_alpha <- sse * sqrt(1/n + mean(x)^2 / ssx)
#t_statistics of intersect
t_alpha <- alpha / se_alpha
p_alpha <- 2 * pt(abs(t_alpha),df = n-2, lower.tail = F)


#or just
summary(fit)$coefficients


#summary(fit) complete output
#'Call:
#'lm(formula = y ~ x)
#'
#'Residuals:
#'    Min      1Q  Median      3Q     Max
#'-7.8050 -1.3661  0.0487  1.6339  5.9264
#'
#'Coefficients:
#'           Estimate Std. Error t value Pr(>|t|)
#'(Intercept) 23.94153    2.81088   8.517   <2e-16 ***
#'x            0.64629    0.04114  15.711   <2e-16 ***
#'---
#'Signif. codes:
#'0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#'
#'Residual standard error: 2.239 on 926 degrees of freedom
#'Multiple R-squared:  0.2105,  Adjusted R-squared:  0.2096
#'F-statistic: 246.8 on 1 (deg of MSSR) and 926 (deg of MSSE)
#’p-value: < 2.2e-16

layout(matrix((c(1,2,3,4)),2))
plot(fit)

fit.influence<- influence.measures(fit)
names(fit.influence)
head(fit.influence$infmat)
par(mfrow=c(1,1))
plot(x,y)
abline(fit,col=2)

newdata=seq(min(x), max(x), 0.01)
prd.interval<-predict(fit, newdata = data.frame(x=newdata), interval =
c("confidence"), level = .9, type="response")
lines(newdata, prd.interval[,2], col = 4) #lower est
lines(newdata, prd.interval[,3], col = 4) #upper est

fit.bigguy <- which(fit.influence$is.inf[,"hat"]==T)
points(x[fit.bigguy],y[fit.bigguy], col=2) #high leverage points


fit.badguy <- which(fit.influence$is.inf[,"cook.d"]==T)
```

```
points(x[fit.badguy],y[fit.badguy], pch="X", col=4) #high cook distance --
outlier

anova(fit)
# Analysis of Variance Table
# Response: y
#            Df Sum Sq Mean Sq F value    Pr(>F)
# x           1 1236.9 1236.93  246.84 < 2.2e-16 ***
# Residuals 926 4640.3    5.01
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1



#Total variation = residual variation + regression variation
#sum(Y_i – Y_bar)^2 =  sum(Y_i – Y_hat)^2 + sum(Y_hat – Y_bar)^2

#Coefficient of Determination
#R-squared = regression variation(SSR) / Total variation(SST=SSR+SSE) = r^2
#Adjusted R-squared = 1 – (n-1)/(n-2)*(1-R-squared)

#F-statistics
#F =  mean regression sum of squares (MSSR) / mean squared error (MSSE)
#   =  (slope of t-statistics )^2

#t-statistics of the slope
#t = r * sqrt(n-2) / sqrt(1-r^2)

#Standard Error of Coef
#var(beta) = standerErrorOfEstimate ^2 / sum(X_i – X_bar)^2
#var(alpha) = standerErrorOfEstimate ^2 * (1/n + X_bar^2 / sum(X_i – X_bar)^2 )

#Standard Error of Prediction (prediction interval at X)
#    standerErrorOfEstimate * sqrt(1 + 1/n + (X – X_bar)^2 / sum(X_i – X_bar)^2
)

#Standard Error of Regression Line (regression line spread at X)
#     standerErrorOfEstimate * sqrt(1/n + (X – X_bar)^2 / sum(X_i – X_bar)^2 )
```







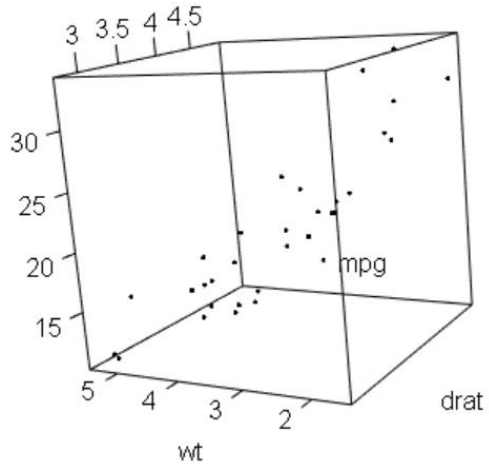


(a) Residuals vs fitted shows correlation. A flat line is better
(b) Standardized residual vs fitted show heteroscedasticity. A flat line is good
(c) Q = mu+q*quantitle. So slope is std error. This detects flat tail
(d) This plot highlights influence of high-leverage cases in deciding the regression line

- Multi-regression

```
attach(mtcars)
model<-lm(mpg~drat+wt)
library(rgl)
plot3d(drat,wt,mpg)

summary(model)

# Call:
# lm(formula = mpg ~ drat + wt)
```

```
# Residuals:
#     Min      1Q  Median      3Q     Max
# -5.4159 -2.0452  0.0136  1.7704  6.7466
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept)   30.290      7.318   4.139 0.000274 ***
# drat           1.442      1.459   0.989 0.330854
# wt            -4.783      0.797  -6.001 1.59e-06 ***
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
# Residual standard error: 3.047 on 29 degrees of freedom
# Multiple R-squared:  0.7609,  Adjusted R-squared:  0.7444
# F-statistic: 46.14 on 2 and 29 DF,  p-value: 9.761e-10

confint(model)
#                 2.5 %    97.5 %
# (Intercept) 15.323629 45.257112
# drat        -1.540615  4.425596
# wt          -6.413010 -3.152770

anova(model)
# Analysis of Variance Table
# Response: mpg
#          Df Sum Sq Mean Sq F value    Pr(>F)
# drat       1 522.48  522.48  56.276 2.871e-08 ***
# wt         1 334.33  334.33  36.010 1.589e-06 ***
# Residuals 29 269.24    9.28
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


# more is fun
lm(mpg~drat+I(wt^2)+wt)
```



- Some generalized regression: logistic regression

```
attach(mtcars)
model<-glm(am~wt, family = "binomial") #need to convert numeric am to factor
```

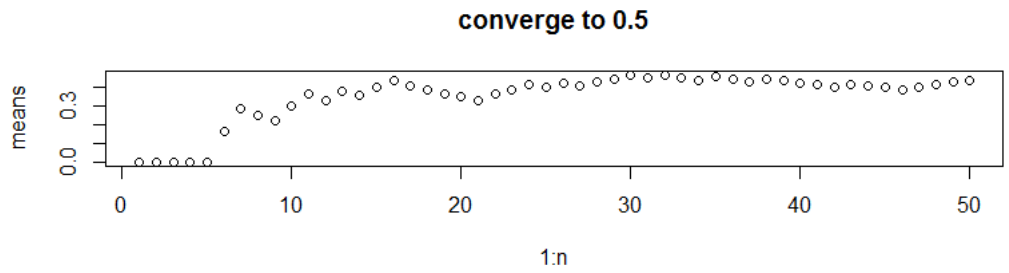
```
types of family
binomial(link = "logit")
gaussian(link = "identity")
Gamma(link = "inverse")
inverse.gaussian(link = "1/mu^2")
poisson(link = "log")
quasi(link = "identity", variance = "constant")
quasibinomial(link = "logit")
quasipoisson(link = "log")

summary(model)
# Call:
# glm(formula = am ~ wt, family = "binomial")
#
# Deviance Residuals:
#      Min        1Q    Median        3Q       Max
# -2.11400  -0.53738  -0.08811   0.26055   2.19931
#
# Coefficients:
#             Estimate Std. Error z value Pr(>|z|)
# (Intercept)   12.040      4.510   2.670  0.00759 **
# wt            -4.024      1.436  -2.801  0.00509 **
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
# (Dispersion parameter for binomial family taken to be 1)
#     Null deviance: 43.230  on 31  degrees of freedom
# Residual deviance: 19.176  on 30  degrees of freedom
# AIC: 23.176
#
# Number of Fisher Scoring iterations: 6

predict(model, newdata = data.frame(wt=1.500), type = "response")
#1
#0.9975382 -> predict to be auto transmission with 99.75% prob
```

4. Limiting Theorem

- LLN

  —mean converges to sample mean

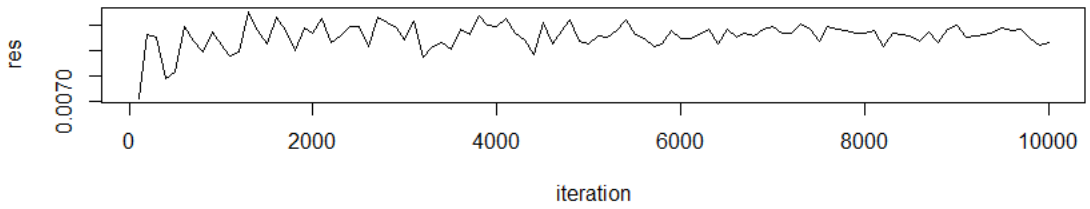
```
n <- 50
means <- cumsum(sample(0:1,n, replace = T))/(1:n)
plot(1:n,means)
title("converge to 0.5")
```



- CLT

  —sample mean is normal disturbed with var/n (n=sample size, var=population variance). If the population var is replaced by sample variance, it is t-distributed.

```
iteration<-seq(100,10000,100)
n<-10
res<-lapply(iteration, function(x)
var(apply(matrix(runif(x*n),x),1,mean)))

plot(iteration, res, 'l')  #converge to (1/12)/10 = 0.008333333
```

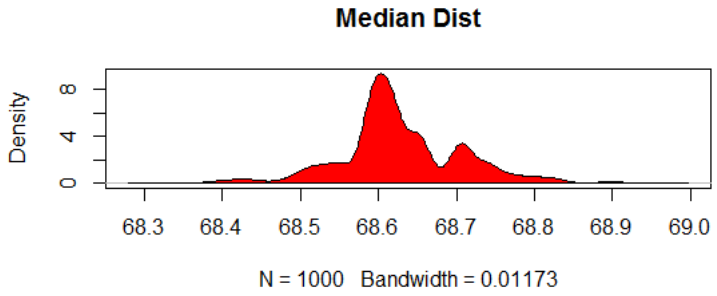


5. Monte Carlo simulation & Resampling(w/o replace), Bootstrapping(w/ replace), permutation test, cross validation

- Bootstrapping

Simulate rv of population without population

```
library(UsingR)
data("father.son")
sonHeight<-father.son$sheight
obser<-length(sonHeight)
boostrapNo<-1000

resample<-matrix(sample(sonHeight, boostrapNo*obser, replace = T),
boostrapNo, obser)
resampleMedian<-apply(resample, 1, median)

d<-density(resampleMedian)
plot(d, main="Median Dist")
polygon(d, col="red")
```



- Permutation Table

```
n <- 100
tr <- rbinom(100, 1, 0.5)
y <- 1 + tr + rnorm(n, 0, 0.3)
diff(by(y, tr, mean))

dist <- replicate(2000, diff(by(y, sample(tr), mean)))
```

- Cross validation, K-fold (no replacement)
  large k => less bias, more variance, small k => more bias, less variance, last one out cross validation

```
library(ggplot2)
diamonds<-diamonds[1:500,]
trainDiamond <- sample(dim(diamonds)[1][1], size = 200)
train<-diamonds[trainDiamond,]

#testing-training cross validation
mylm <- lm(data = train, x ~ y + z)
mean((diamonds$x-predict(mylm, diamonds))[-trainDiamond]^2)
#0.002657964

#Leave one out cross validation
myglm <- glm(data = train, x ~ y + z)
library(boot)
cv.glm(data = train, myglm)$delta
#0.002852013(raw) 0.002851553(adjust)

#k-fold cross validation
cv.glm(data = train, myglm,K = 10)$delta
#0.002789518(raw) 0.002783287(adjust)
```

6. Standard errors of estimates, Bootstrapping standard errors

$$Err_{.632} = .368 \cdot \overline{err} + .632 \cdot Err_{boot(1)}, where\ \overline{err} = \frac{1}{N}\sum_{i}^{N} L(y_i, f(x_i))$$

$$se(\hat{\mu}_i) = SD(\hat{\mu}_i) = \sigma_i/\sqrt{T}$$

$$se\left(\hat{\sigma}_i{}^2\right) \approx \frac{\sqrt{2}\sigma_i^2}{\sqrt{T}} = \frac{\sigma_i^2}{\sqrt{\frac{T}{2}}}, se(\hat{\sigma}_i) \approx \frac{\sigma_i}{\sqrt{2}\sqrt{T}}, se\left(\rho_{ij}\right) \approx \frac{(1-\rho_{ij}^2)}{\sqrt{T}}$$

for multi-asset random walk simulation.

$$\widehat{SE}_{boot}(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_j^B \left(\hat{\theta}_j^* - \frac{1}{B}\sum_j^B \hat{\theta}_j^*\right)^2}$$

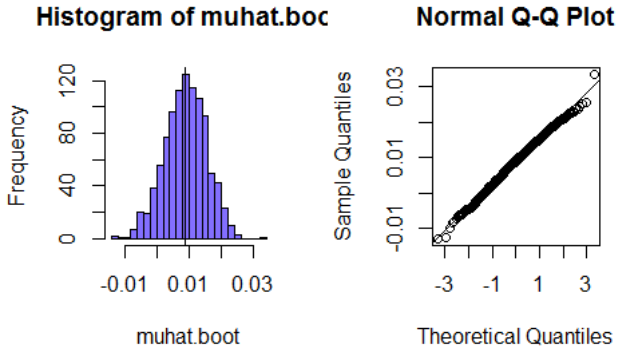
Bootstrap confidence interval: $[q^*_{.025}, q^*_{.925}]$

- Boot package

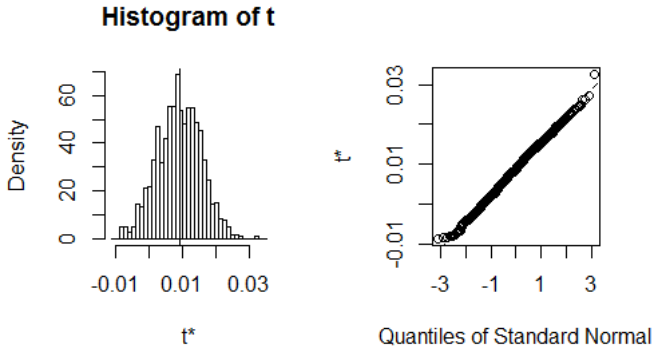
```
library(boot)
library(zoo)

end = Sys.Date()
d<- as.POSIXlt(as.Date(end))
d$year <- d$year - 10
start = as.Date(d)
par(mfrow=c(1,1))
#monthly adj closed price 10 year
MSFT.prices = get.hist.quote(instrument="MSFT", start=start,
                             end=end, quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")


MSFT = diff(log(MSFT.prices))

# random sample from MSFT return <- brutal force
B = 999
muhat.boot = rep(0, B)
nobs = length(MSFT)
for (i in 1:B) {
      boot.data = sample(MSFT, nobs, replace=TRUE)
      muhat.boot[i] = mean(boot.data)
}

# bootstrap bias
mean(muhat.boot) - mean(MSFT) # 3.269e-05
# bootstrap SE
sd(muhat.boot) # 0.006427
# analytic SE
sd(MSFT)/sqrt(length(MSFT)) # 0.006449


par(mfrow=c(1,2),oma=c(0,0,1.5,0))
hist(muhat.boot, col="slateblue1", breaks=30)
abline(v=mean(MSFT), col="white", lwd=2)
qqnorm(muhat.boot)
qqline(muhat.boot)
title("Boot w/o boot function", outer=TRUE)
```

```
################## use boot ##############

##Mean##
mean.boot = function(x, idx) {
      ans = mean(x[idx])
      ans
}

MSFT.mean.boot = boot(MSFT, statistic = mean.boot, R=999)
MSFT.mean.boot
# Bootstrap Statistics :
#     original     bias    std. error
# t1* 0.008874 -1.459e-06    0.006584

plot(MSFT.mean.boot)
title("Boot w/ boot function", outer=TRUE)
boot.ci(MSFT.mean.boot, conf = 0.95, type = c("norm","perc"))
#BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
#Based on 999 bootstrap replicates
# CALL :
# boot.ci(boot.out = MSFT.mean.boot, conf = 0.95, type = c("norm",
#     "perc"))
# Intervals :
# Level      Normal              Percentile
# 95%   (-0.0040,  0.0218 )   (-0.0040,  0.0215 )
# Calculations and Intervals on Original Scale

##sd##
sd.boot = function(x, idx) {
      ans = sd(x[idx])
      ans
}
MSFT.sd.boot = boot(MSFT, statistic = sd.boot, R=999)
MSFT.sd.boot
plot(MSFT.sd.boot)
title("Boot sd", outer=TRUE)
boot.ci(MSFT.sd.boot, conf=0.95, type=c("norm", "basic", "perc"))


##Value-at-Risk
ValueAtRisk.boot = function(x, idx, p=0.05, w=100000) {
      q = mean(x[idx]) + sd(x[idx])*qnorm(p)
      VaR = (exp(q) - 1)*w
      VaR
}

MSFT.VaR.boot = boot(MSFT, statistic = ValueAtRisk.boot, R=999)
MSFT.VaR.boot
boot.ci(MSFT.VaR.boot, conf=0.95, type=c("norm", "perc"))

plot(MSFT.VaR.boot)
title("Boot VaR", outer=TRUE)
```

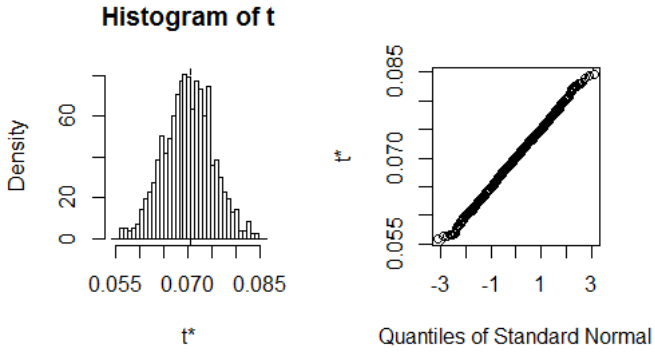
Boot w/o boot function

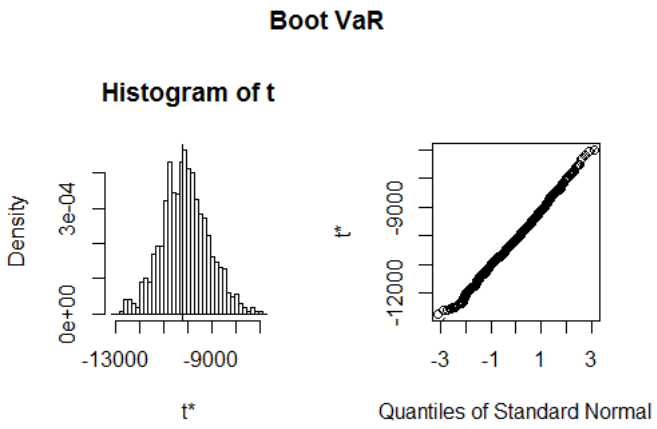


Boot w/ boot function



Boot sd





7. Hypothesis testing, confidence interval

- Binomial Confidence interval

```
#95% confidence

binom.test(55,100, p = 0.5)$conf
#0.4472802 0.6496798 <- exact interval of being success

#approx
0.55 + c(-1,1)*qnorm(0.975)*sqrt(.55*.45/100)
#0.452493 0.647507
```

- Poisson Confidence interval

```
#say during one hour window an event happens
#6 times on average. What is the 95% conf inv
#for such event to happen in given 1 min?

poisson.test(6,60, r = 0.1)$conf
#0.03669824 0.21765790 <-exact interval

#approx.
6/60 + c(-1,1)*qnorm(0.975)*sqrt(6/60/60)
#0.01998481 0.18001519
```

- T-Test

```
library(UsingR)
data("father.son")

#null hypothesis: father height == son height
t.test(father.son$sheight-father.son$fheight, mu = 0)

#same as
t.test(father.son$sheight,father.son$fheight,mu = 0, paired = T)

#output
#'t = 11.789, df = 1077, p-value < 2.2e-16
#'alternative hypothesis: true mean is not equal to 0
```

```
#'95 percent confidence interval:
#' 0.8310296 1.1629160 (unit: inch)
#'mean of x : 0.9969728
#'so we conclude son taller than father
```

- Power

  $power = P(reject\ H_0 | H_1\ is\ true)$. If population mean ($\mu_0$) is not equal to sample mean (μ), define noncentrality parameter,

$$\delta = \frac{(\mu_0 - \mu)}{\sqrt{n}\sigma}$$

noncentral t distribution.

```
 alpha <- 0.05
z <- qnorm(1-alpha)

mu_0 <- 12   #alterntive hypothesis population mean
mu <- 10     #sample mean
sigma <- 5   #population var
n <- 10      #sample size
delta <- (mu_0-mu)/sigma*sqrt(n)

power.t.test(n = n, delta=delta, sd =sigma, type = "one.sample",
alternative = "one.sided")
#or
pt(mu_0-mu,df = n, ncp = delta, lower.tail = FALSE)
```

8. Experimental design, inferential statistics

- Maximum Likelihood, entropy: $S = p \ln p$

```
MaxLik <- function(data){
      function(para){
            mu <- para[1]
            sigma <- para[2]
            a <- -length(data)*log(2*pi*sigma^2)/2
            b <- -sum((data-mu)^2)/(sigma^2)/2
            -(a + b)
      }
}

normals <- rnorm(100,1,2)

optim(c(mu = 0, sigma = 1),f=MaxLik(normals))$par
```

- Multivariable Likelihood function -> maximize log normal

```
loglike.normal = function(theta, x) {
```

```
      # theta   parameters c(mu,sig2)
      # x       vector of data
      mu = theta[1]
      sig2 = theta[2]
      n = length(x)
      a1 = -(n/2)*log(2*pi)-(n/2)*log(sig2)
      a2 = -1/(2*sig2)
      y = (x-mu)^2
      ans = a1+a2*sum(y)
      # return -1 * loglike because multi optim function only do min
      return(-ans)
}

n = 50
set.seed(123)
x = rnorm(n, mean=0, sd=1)
theta.start = c(0,1)
ans = optim(par=theta.start, fn=loglike.normal, x=x,
            method="BFGS")
ans$par
#0.03441726 0.84010990

mean(x)  #0.03441726
var(x)*(n-1)/nc #0.84010990

ans = optim(par=theta.start, fn=loglike.normal, x=x,
            method="BFGS", hessian=TRUE)
names(ans)
ans$hessian
se.mle = sqrt(diag(solve(ans$hessian)))
se.mle
#0.1296233 0.1680251

#use the maxLik function
library(maxLik)
loglike.normal = function(theta, x) {
      mu = theta[1]
      sig2 = theta[2]
      n = length(x)
      a1 = -(n/2)*log(2*pi)-(n/2)*log(sig2)
      a2 = -1/(2*sig2)
      y = (x-mu)^2
      ans = a1+a2*sum(y)
      return(ans)
}
theta.start = c(0,1)
names(theta.start) = c("mu","sig2")
theta.mle = maxLik(loglike.normal, start=theta.start, x=x)
class(theta.mle)
names(theta.mle)
theta.mle
summary(theta.mle)
# Maximum Likelihood estimation
# Newton-Raphson maximisation, 5 iterations
# Return code 1: gradient close to zero
```

```
# Log-Likelihood: -66.59079
# 2  free parameters
# Estimates:
#      Estimate Std. error t value  Pr(> t)
# mu     0.0344     0.1296   0.265    0.791
# sig2   0.8401     0.1680   4.999 5.75e-07 ***
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

- One can thing regress as maximize the likehood function

$$L(\beta, \sigma) = P(Y \mid X, \beta, \sigma)$$ where σ is the var in the noise

- Test proportion of success (z-test)

```
totalnum = 1000
success = 400

# H_0: P(success) is equal to 0.5
prop.test(x=success, n=totalnum, alternative = "two.side")

# 1-sample proportions test with continuity correction
# data:  success out of totalnum, null probability 0.5
# X-squared = 39.601, df = 1, p-value = 3.115e-10  "<- very small p"
# alternative hypothesis: true p is not equal to 0.5
# 95 percent confidence interval:
# 0.3695830 0.4311949
# sample estimates:
# p
# 0.4


#H0: P(success) is less than 0.5
prop.test(x=yes, n=totalnum, alternative = "greater")
```

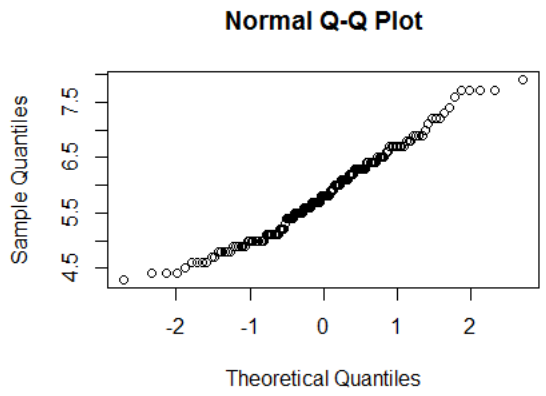
- test normality

```
leng <- iris$Sepal.Length

jarque.bera.test(leng)
#   Jarque Bera Test
#data:  leng
#X-squared = 4.4859, df = 2, p-value = 0.1061
```

$$JB = \frac{n-k+1}{6}\left(S^2 + \frac{1}{4}(C-3)^2\right)$$, S = skewness, C = kurtosis

```
# by hand
leng.skew <- skewness(leng)
leng.esskurt <- kurtosis(leng)
n= length(leng)

n*(leng.skew^2+(leng.esskurt^2)/4)/6
```

```r
# 4.486

1-pchisq(4.4859,df=2)
#p-value 0.1061


shapiro.test(leng) #only accept 5000 argument. For large dataset, take a
random subset

# Shapiro-Wilk normality test
# data:  iris$Sepal.Length
# W = 0.97609, p-value = 0.01018

library(nortest)
ad.test(leng)

# Anderson-Darling normality test
# data:  iris$Sepal.Length
# A = 0.8892, p-value = 0.02251

library(nortest)
cvm.test(leng)

# Cramer-von Mises normality test
# data:  leng
# W = 0.1274, p-value = 0.04706

lillie.test(leng)

# Lilliefors (Kolmogorov-Smirnov) normality test
# data:  leng
# D = 0.088654, p-value = 0.005788

sf.test(leng)

# Shapiro-Francia normality test
# data:  leng
# W = 0.97961, p-value = 0.02621

pearson.test(leng)

# Pearson chi-square normality test
# data:  leng
# P = 17.4, p-value = 0.1352

# Last but not least a graphical test. If its perfect normal, it'll be a
straight line
qqnorm(leng)
```



- Test Correlation, and autocorrelation

$$se(\rho_{ij}) \approx \frac{(1-\rho_{ij}^2)}{\sqrt{T}}$$

→ if $\bar{\rho} = 0, se(\rho) = \frac{1}{\sqrt{T}}$, hence the test-statistic is

$$(\rho_i - 0) \,/\, se(\rho_{ij}) = \sqrt{T}\,\rho_i$$

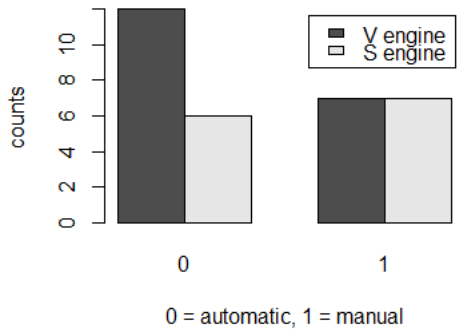
In other words for 95% CI, if $|\rho_i| > 2/\sqrt{T}$, reject H0: no correlation

```
cor( Sepal.Length, Sepal.Width, method = "pearson")
cor( Sepal.Length, Sepal.Width, method = "spearman")
cor( Sepal.Length, Sepal.Width, method = "kendall")

cor.test(Sepal.Length,Sepal.Width, method="pearson")
#   Pearson's product-moment correlation
# data:  Sepal.Length and Sepal.Width
# t = -1.4403, df = 148, p-value = 0.1519
# alternative hypothesis: true correlation is not equal to 0
# 95 percent confidence interval:
#  -0.27269325  0.04351158
# sample estimates:
#        cor
# -0.1175698
```

- Test independency: chi-squared test, fisher's (non-parametric)

```
attach(mtcars)

#contingency table -> table(am,vs)
barplot(table(am,vs), beside = T,
        legend.text = c("V engine","S engine"),
        xlab = "0 = automatic, 1 = manual", ylab = "counts")

chisq.test(table(am,vs))

# Pearson's Chi-squared test with Yates' continuity correction
# data:  table(am, vs)
# X-squared = 0.34754, df = 1, p-value = 0.5555
```

```r
#for non-parametric data use Fisher

fisher.test(table(am,vs))

# Fisher's Exact Test for Count Data

# data:  table(am, vs)
# p-value = 0.4727
# alternative hypothesis: true odds ratio is not equal to 1
# 95 percent confidence interval:
# 0.3825342 10.5916087
# sample estimates:
# odds ratio
# 1.956055
```



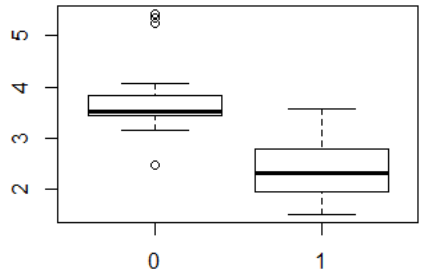
- Test population mean (1 sample)

```r
#H0: mean is 6 or higher
t.test(x = leng, mu = 6, alternative = "less", conf.level = 0.95)

#H0: mean is 6 or smaller
t.test(x = leng, mu = 6, alternative = "greater", conf.level = 0.95)

#H0: mean is 6
t.test(x = leng, mu = 6, alternative = "two.side", conf.level = 0.95)
```

- Test 2 samples mean
  (data is normal): t-test or called Welch two-sample test

```r
attach(mtcars)
boxplot(data=mtcars, wt~am)

#check variance
var(wt[am==0]); var(wt[am==1]) # this justifies to var.equal = F below

#H_0: the mean weigth of automatic cars is same as the mean weight of
manual cars
t.test(wt~am, alternative = "two.sided",
```

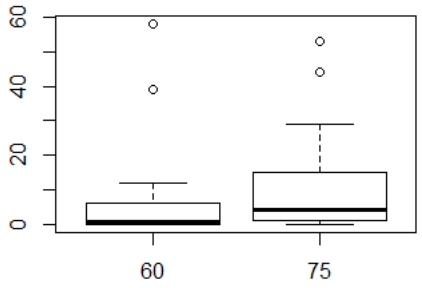
```
        conf.level = 0.95, mu = 0, paired = F, var.equal = F)

#same as
t.test(wt[am==0], wt[am==1])


# Welch Two Sample t-test
# data:  wt by am
# t = 5.4939, df = 29.234, p-value = 6.272e-06
# alternative hypothesis: true difference in means is not equal to 0
# 95 percent confidence interval:
      # 0.8525632 1.8632262
# sample estimates:
      # mean in group 0 mean in group 1
# 3.768895        2.411000
```



- Test 2 samples mean
  (data is not normal, called non-parametric): Mann Whitney U test

```
library(MASS)
attach(ships)
boxplot(incidents~period)

wilcox.test(incidents~period, mu=0, alt="two.side"
            , conf.int=T, conf.level = .95, paired=F
            , correct=T, exact=F)

# Wilcoxon rank sum test with continuity correction
# data:  incidents by period
# W = 140.5, p-value = 0.1024
# alternative hypothesis: true location shift is not equal to 0
# 95 percent confidence interval:
# -7.999965e+00  4.390335e-05
# sample estimates:
# difference in location
# -1.000044


wilcox.test(incidents~period, mu=0, alt="two.side"
            , conf.int=T, conf.level = .95, paired=T
```

```
            , correct=F, exact=F)

# Wilcoxon signed rank test
# data:  incidents by period
# V = 22.5, p-value = 0.05897
# alternative hypothesis: true location shift is not equal to 0
# 95 percent confidence interval:
# -11.0000050   0.4999812
# sample estimates:
# (pseudo)median
# -4.000042
```

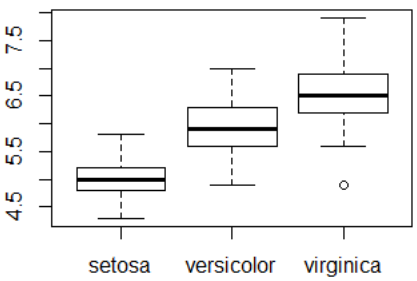


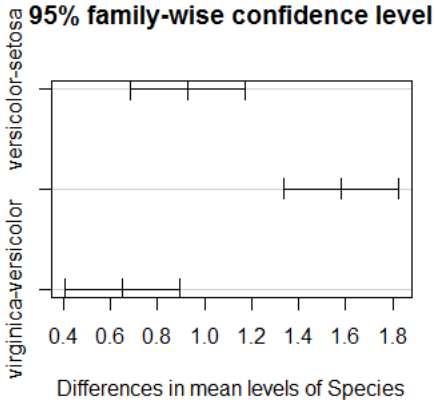
- Test >2 samples mean

(data is normal): F, ANOVA— also used in multi-regression

```
attach(iris)
levels(Species)
boxplot(Sepal.Length~Species)
by(Sepal.Length, Species, mean)

oneway.test(Sepal.Length~Species) #F-test

# One-way analysis of means (not assuming equal variances)
# data:  Sepal.Length and Species
# F = 138.91, num df = 2.000, denom df = 92.211, p-value < 2.2e-16


irisANOVA = aov(Sepal.Length~Species, data = iris); irisANOVA
# Terms:
                # Species Residuals
# Sum of Squares  63.21213  38.95620
# Deg. of Freedom        2       147

# Residual standard error: 0.5147894
# Estimated effects may be unbalanced
0
coefficients(irisANOVA)
# (Intercept) Speciesversicolor  Speciesvirginica
#       5.006            0.930            1.582
```

```
TukeyHSD(irisANOVA)

# Tukey multiple comparisons of means
# 95% family-wise confidence level
# Fit: aov(formula = Sepal.Length ~ Species, data = iris)
# $Species
#                       diff       lwr       upr p adj
# versicolor-setosa    0.930 0.6862273 1.1737727     0
# virginica-setosa     1.582 1.3382273 1.8257727     0
# virginica-versicolor 0.652 0.4082273 0.8957727     0
plot(TukeyHSD(irisANOVA))

#similarly
library(DTK)
TK.test( x= Sepal.Length, f=Species)

# Tukey multiple comparisons of means
# 95% family-wise confidence level
# Fit: aov(formula = x ~ f)
# $f
#             diff       lwr       upr p adj
# versicolor-setosa    0.930 0.6862273 1.1737727     0
# virginica-setosa     1.582 1.3382273 1.8257727     0
# virginica-versicolor 0.652 0.4082273 0.8957727     0


library(car)
leveneTest(Sepal.Length, Species, data=iris, center = "mean")

# Levene's Test for Homogeneity of Variance (center = "mean": iris)
#         Df F value    Pr(>F)
# group   2  7.3811 0.0008818 ***
#       147
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```





- Test >2 samples mean

(data is not normal, non-parametric): Kruskal Wallis tests

```r
kruskal.test(data = iris, Sepal.Length~Species)

# Kruskal-Wallis rank sum test
# data:  Sepal.Length by Species
# Kruskal-Wallis chi-squared = 96.937, df = 2, p-value < 2.2e-16

#alternatively use t.test with a p adjust correction, e.g. Holm–Bonferroni
method

pairwise.t.test(x=Sepal.Length, g=Species, p.adjust.method = "BH")

# Pairwise comparisons using t tests with pooled SD
# data:  Sepal.Length and Species
# setosa  versicolor
# versicolor 1.3e-15 -
# virginica  < 2e-16 2.8e-09
# P value adjustment method: BH
```

Time Series Analysis

9. Stationarity, seasonality

- Non-constant mean (trend) => non- stationary, heteroscedasticity
- Unit root test: augmented Dickey–Fuller test.

Stationary : time series is invariance under time translation

$$X_t = \phi \cdot X_{t-1} + WN(0, \sigma^2)$$

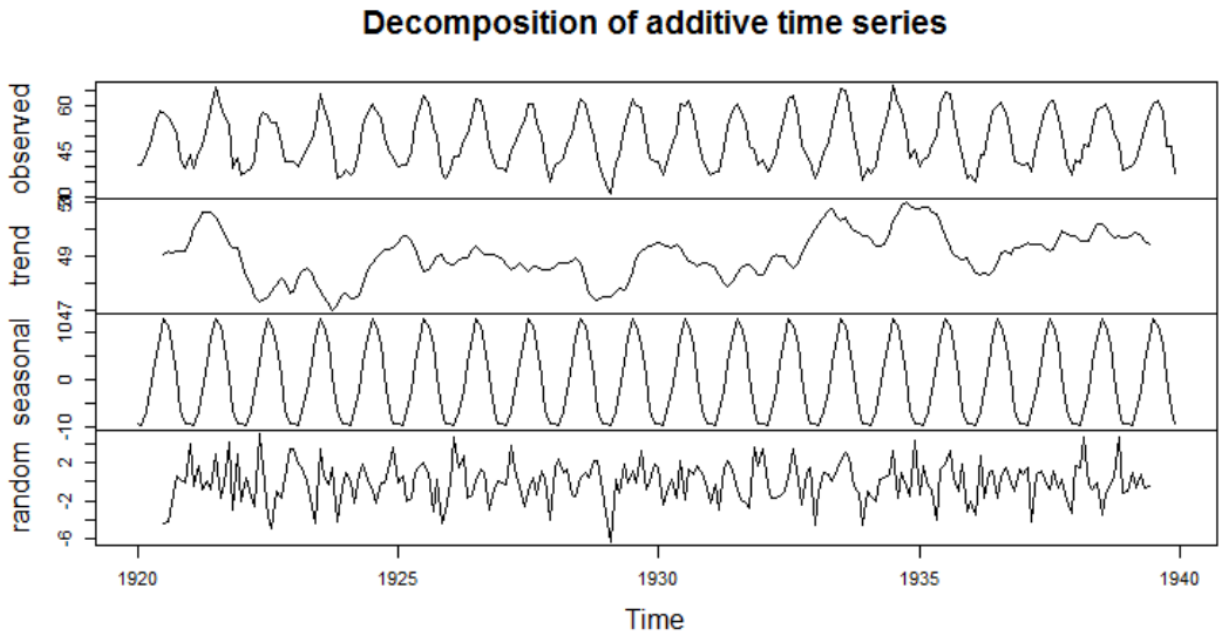
H0: $\phi = 1$ (unit root, non-stationary) ϕ is root of polynomial of $AR(p)$

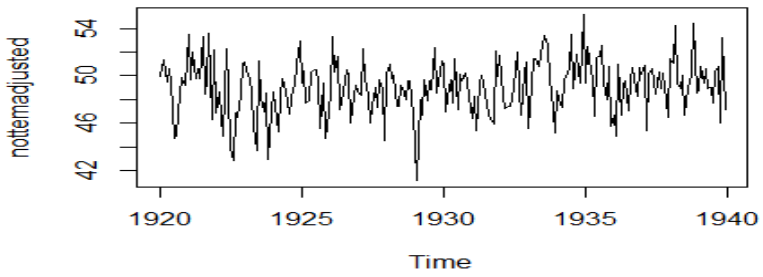
$1 - lag_1 \cdot \phi - lag_2 \cdot \phi^2 - \ldots$

H0: $|\phi| < 1$

```r
library(tseries)
plot(nottem) # the nottem dataset
plot(decompose(nottem, type = "additive"))
adf.test(nottem)

# Augmented Dickey-Fuller Test
# data:  nottem
# Dickey-Fuller = -12.998, Lag order = 6,
# p-value = 0.01
# alternative hypothesis: stationary


x <- diffinv(nottem) # non-stationary
plot(x)
adf.test(x)
```

```
# Augmented Dickey-Fuller Test
# data:  x
# Dickey-Fuller = -0.63448, Lag order
# = 6, p-value = 0.9752
# alternative hypothesis: stationary


# seasonal adjustment
nottemadjusted = nottem - decompose(nottem, "additive")$seasonal
plot(nottemadjusted)
```

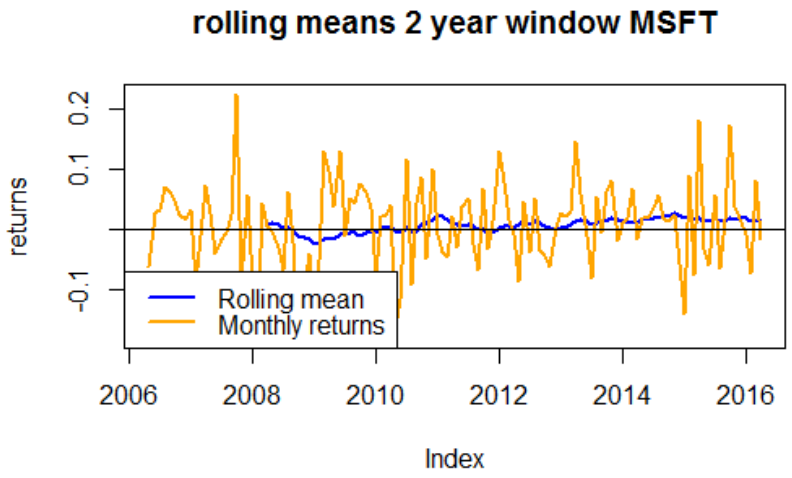


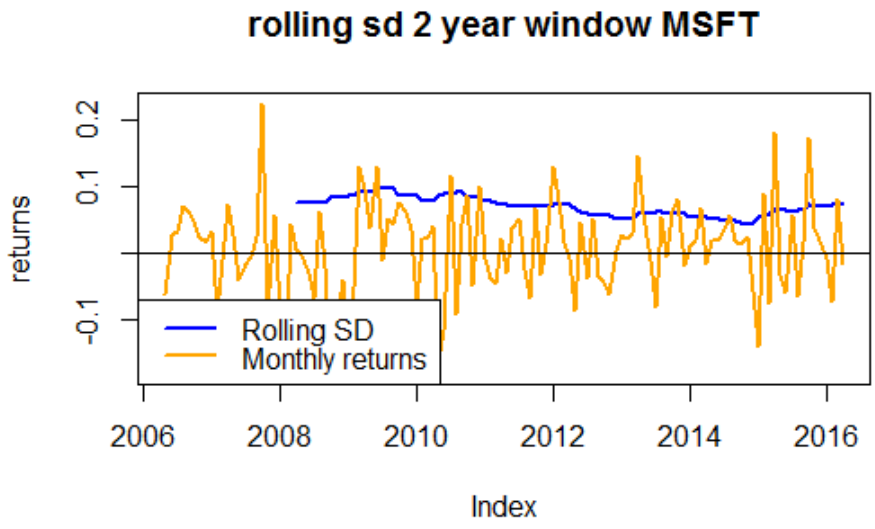


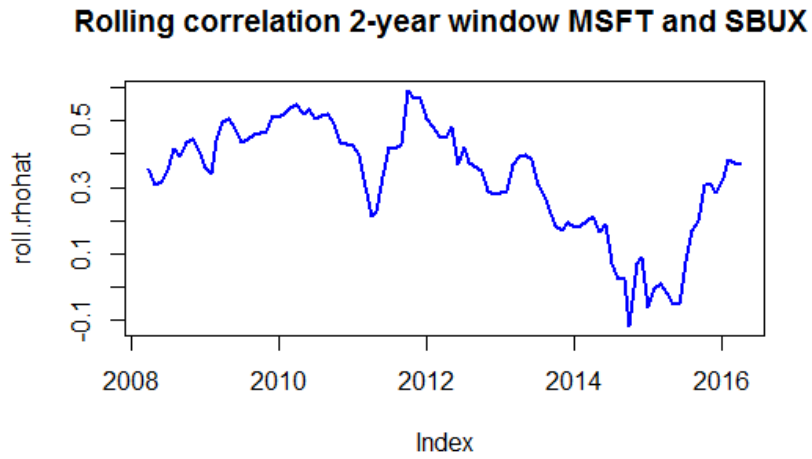
- Rolling Means; Rolling Sd; Rolling Cov / Cor

```
#
library(zoo)
end = Sys.Date()
d<- as.POSIXlt(as.Date(end))
```

```
d$year <- d$year - 10
start = as.Date(d)
par(mfrow=c(1,1))

MSFT.prices = get.hist.quote(instrument="msft", start=start,
                             end=end, quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")
MSFT = diff(log(MSFT.prices))


SBUX.prices = get.hist.quote(instrument="sbux", start=start,
                             end=end, quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")
SBUX = diff(log(SBUX.prices))

roll.muhat = rollapply(MSFT, width=24,
                       FUN=mean, align="right")
plot(merge(roll.muhat,MSFT), plot.type="single",
     main="rolling means 2-year MSFT",ylab="returns",
     lwd=c(2,2), col=c("blue","orange"))
abline(h=0)
legend(x="bottomleft",legend=c("Rolling mean","Monthly returns"),
       lwd=c(2,2), col=c("blue","orange"))

rhohat = function(x){
      cor(x)[1,2]
}
roll.rhohat = rollapply(merge(MSFT,SBUX), width=24,
                        FUN=rhohat,by.column = F, align="right")

plot(roll.rhohat, lwd=2, col=4, main="Rolling cor 2-year window MSFT and SBUX
")
```







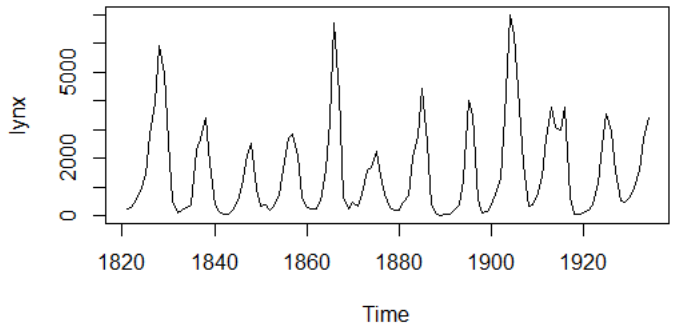
Hence the assumptions of CER model: constant return, constant sd are correct; but constant correlation is not.

10. Autocorrelation

- Stationarity process: $var(Y_t) = var(Y_{t-j})$
  - Unit Test
  - Durbin Watson test

```
library(lmtest)
plot(lynx)
dwtest(lynx[-114] ~ lynx[-1]) #1 lag time difference,
# autocorrelation of 1st order, not robust

# Durbin-Watson test
# data:  lynx[-114] ~ lynx[-1]
# DW = 1.1296, p-value = 1.148e-06
```

```
# alternative hypothesis: true autocorrelation is greater than 0

#in comparison to random normal, not auto correlated
x = rnorm(700)
dwtest(x[-700] ~ x[-1])

# Durbin-Watson test
# data:  x[-700] ~ x[-1]
# DW = 2.0056, p-value = 0.5302
# alternative hypothesis: true autocorrelation is greater than 0
```



- Autocorrelation function (ACF)
- Partial autocorrelation function (PACF)

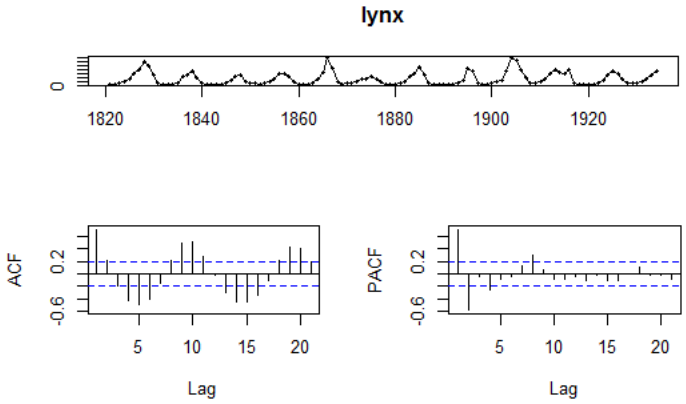
ACF=> function of $corr(Y_t, Y_{t-j})$ of j

PACF=> coef of $lm(Y_t \sim Y_{t-1} + Y_{t-2} + \cdots)$

```
lynx.acf<-acf(lynx, lag.max = 10, plot = F )
lynx.pacf<-pacf(lynx, lag.max = 10, plot = F)

library(forecast)
tsdisplay(lynx)

acf_pacf_tab=cbind(as.matrix(lynx.acf$acf[-1]),lynx.pacf$acf)
dimnames(acf_pacf_tab)=list(c(1:10),c("acf","pacf"))
print(acf_pacf_tab)

#           acf         pacf
# 1   0.7108187  0.71081868
# 2   0.2144115 -0.58789184
# 3  -0.1885254 -0.03906685
# 4  -0.4334992 -0.24956946
# 5  -0.5022176 -0.09437599
# 6  -0.4003496 -0.05207440
# 7  -0.1479847  0.11884341
# 8   0.2183651  0.30121848
```

```
# 9   0.5009080  0.05457031
# 10  0.5139073 -0.08115986


#now we will verify these numbers
library(dplyr)
for (k in c(1:10)){
      assign(paste("lynx.lag", k, sep = ""), lag(lynx, n=k))
}
lynx.lag3

#check ACF(k=3)
k=3
cor(lynx[!is.na(lynx.lag3)], na.omit(lynx.lag3))
#-0.1911877  close to -0.1885254, not exactly same, because R uses boostrap to
do acf

fit<-lm(lynx~lynx.lag1+lynx.lag2+lynx.lag3+lynx.lag4+lynx.lag5
        +lynx.lag6+lynx.lag7+lynx.lag8+lynx.lag9+lynx.lag10)
summary(fit)

# Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
# (Intercept) 594.693979 243.678635   2.440   0.0166 *
# lynx.lag1     0.991569   0.103502   9.580 1.58e-15 *** <-only two are signf
# lynx.lag2    -0.589400   0.144820  -4.070 9.86e-05 *** <-close to pacf above
# lynx.lag3     0.209098   0.153731   1.360   0.1771
# lynx.lag4    -0.155269   0.152983  -1.015   0.3128
# lynx.lag5     0.007121   0.152731   0.047   0.9629
# lynx.lag6    -0.011158   0.152491  -0.073   0.9418
# lynx.lag7    -0.137568   0.151698  -0.907   0.3668
# lynx.lag8     0.172092   0.151054   1.139   0.2575
# lynx.lag9     0.168695   0.140362   1.202   0.2325
# lynx.lag10   -0.060023   0.098670  -0.608   0.5445
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
# Residual standard error: 814.1 on 93 degrees of freedom
#   (10 observations deleted due to missingness)
# Multiple R-squared:  0.7427,  Adjusted R-squared:  0.7151
# F-statistic: 26.85 on 10 and 93 DF,  p-value: < 2.2e-16


#or use
ar(lynx, aic = F, order.max = 10)
# Coefficients:
#       1        2        3        4        5        6        7        8
#  1.0259  -0.5753   0.1760  -0.1392  -0.0141  -0.0019  -0.1561   0.1963
#       9       10
#  0.1375  -0.0812
# Order selected 10  sigma^2 estimated as  733939


#we can use lag1,2 coef to find the seasonality
phi1<-fit$coefficients["lynx.lag1"]
phi2<-fit$coefficients["lynx.lag2"]
```

```
season <- 2*pi / acos(abs(phi1)/2/sqrt(abs(phi2)))
print(as.numeric(season))
#7.232481  shows 7-month cycle


#how do we know k=10 for lm above is the right choice
lynx.ar = ar(lynx)
lynx.ar$order
#8 has the minimun aic statistic

lynx.ar$order.max
#20 is the maximum

ar(lynx, aic = T)
# Coefficients:
#       1        2        3        4        5        6        7        8
#  1.0379  -0.6063   0.1910  -0.1411  -0.0207   0.0199  -0.2046   0.3012
# Order selected 8  sigma^2 estimated as  726897


#Now manually check unit test
roots <- polyroot(c(1,-1*lynx.ar$ar))
sqrt(Conj(roots)*roots)
# [1] 1.046849+0i 1.212855+0i 1.212855+0i 1.046849+0i 1.139951+0i 1.315036+0i
1.139951+0i
# [8] 1.205099+0i
# all >1, The model is consistent. Its indeed stationary
```



11. Multi-Time Series: Vector Autoregressive (VAR) model

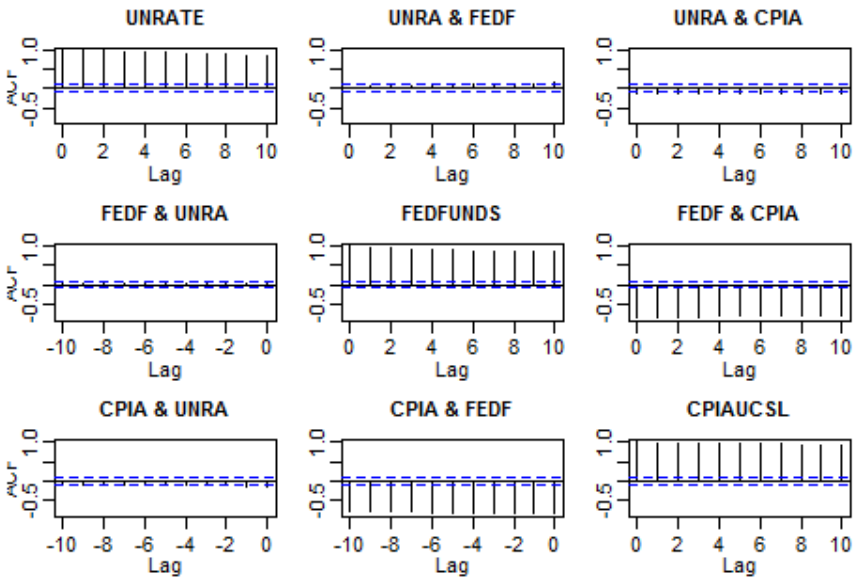
- Macroeconomic model

courtesy of http://ocw.mit.edu/courses/mathematics/18-s096-topics-in-mathematics-with-applications-in-finance-fall-2013/case-studies/fm_casestudy_fx_1.r

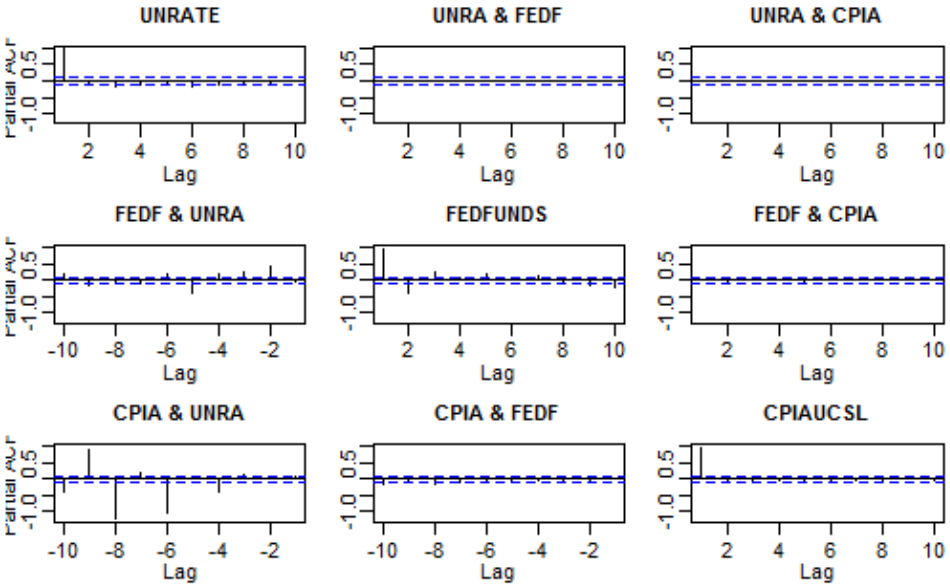
```
#Macro Variables (rates are monthly if not specified)
#https://research.stlouisfed.org/fred2/series/+
#           Season Adj  Desc
#UNRATE     Yes         Civilian Unemployment
```

```
#FEDFUNDS    No          Effective Federal Funds
#TB3MS       No          3-Month Treasury Bill: Secondary Market
#CPIAUCSL    Yes         Consumer Price Index for All Urban Consumers
#M1SL        Yes         M1 Money Stock
#GDPDEF      Yes         (Quarterly) GDP Price Deflator
#GDP         Yes         (Quarterly*4=annualized)GDP
#GPDI        Yes         (annualized)Gross Private Domestic Investment
#TWEXBMTH    No          Trade Weighted U.S. Dollar
#SP500       No          (daily) S&P 500


library(quantmod)
getSymbols(c("UNRATE",  "FEDFUNDS",   "TB3MS",
             "CPIAUCSL",  "M1SL",    "GDPDEF",
             "GDP",       "GPDI",    "TWEXBMTH",
             "SP500"),src="FRED")

ymat<-window(merge(UNRATE,FEDFUNDS,CPIAUCSL),start = as.Date("1980-01-01"), end
= as.Date("2015-12-31"))
head(ymat)
acf(ymat, lag.max = 10)

acf(ymat,type = "partial", lag.max = 10)

library(vars)
ymat.var <- VARselect(ymat, lag.max = 20, type = "const")
#knowing what order we need
ymat.var$selection
# AIC(n)  HQ(n)  SC(n) FPE(n)
#     14      5      2     14
ymat.var.BIC <- VAR(ymat,ic = "SC", p = ymat.var$selection["SC(n)"], type =
"const")

ymat.var.BIC
#VAR Estimation Results:
#=======================
#
# Estimated coefficients for equation UNRATE:
# ===========================================
# Call:
# UNRATE = UNRATE.l1 + FEDFUNDS.l1 + CPIAUCSL.l1 + UNRATE.l2 + FEDFUNDS.l2 +
#CPIAUCSL.l2 + const
#
#   UNRATE.l1 FEDFUNDS.l1 CPIAUCSL.l1   UNRATE.l2 FEDFUNDS.l2 CPIAUCSL.l2
#  1.12266019 -0.01095739 -0.01499979 -0.12659643  0.02514381  0.01595799
#       const
# -0.19957072
#
#
# Estimated coefficients for equation FEDFUNDS:
# =============================================
# Call:
# FEDFUNDS = UNRATE.l1 + FEDFUNDS.l1 + CPIAUCSL.l1 + UNRATE.l2 + FEDFUNDS.l2 +
#CPIAUCSL.l2 + const
#
```

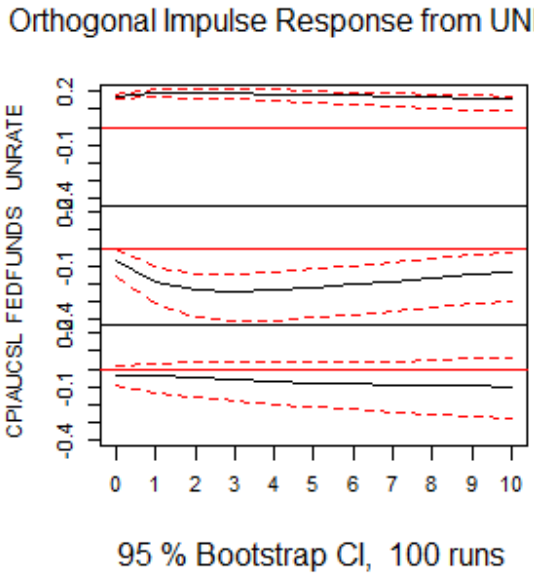
```
#    UNRATE.l1  FEDFUNDS.l1  CPIAUCSL.l1    UNRATE.l2  FEDFUNDS.l2  CPIAUCSL.l2
# -0.553171076  1.326596760  0.006339352  0.539818262 -0.380501513 -0.010356311
#        const
#  0.985321303
#
# Estimated coefficients for equation CPIAUCSL:
# =============================================
# Call:
# CPIAUCSL = UNRATE.l1 + FEDFUNDS.l1 + CPIAUCSL.l1 + UNRATE.l2 + FEDFUNDS.l2 +
CPIAUCSL.l2 + const
#
#   UNRATE.l1 FEDFUNDS.l1 CPIAUCSL.l1   UNRATE.l2 FEDFUNDS.l2 CPIAUCSL.l2
#  0.09685896  0.07807028  1.42170217 -0.10083232 -0.06539011 -0.42128779
#       const
#  0.10948202


plot(irf(ymat.var.BIC))
#impulse: meaning forecast for all variables while changing one variable
#because of vector autocorrelation, all variables are intertwined to each other
```



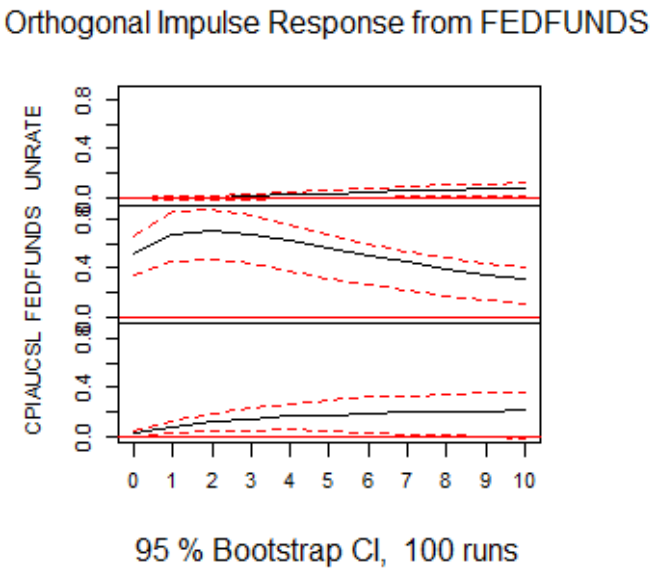
ACF



PACF



Raising unemployment rate -> lower fed funds rate

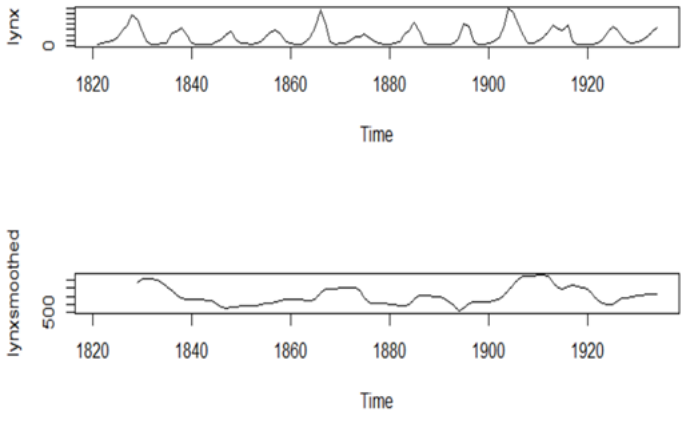


Initial increase in Fed rate -> modest increase in unemployment rate (i.e. unrate persists).
Despite Fed effort to control inflation -> CPI raising

12. Forecasting models & smoothing

- Simple moving average

```
library("TTR")

lynxsmoothed = SMA(lynx, n = 9); 9th order

par(mfrow=c(2,1))
plot(lynx)
plot(lynxsmoothed)
```

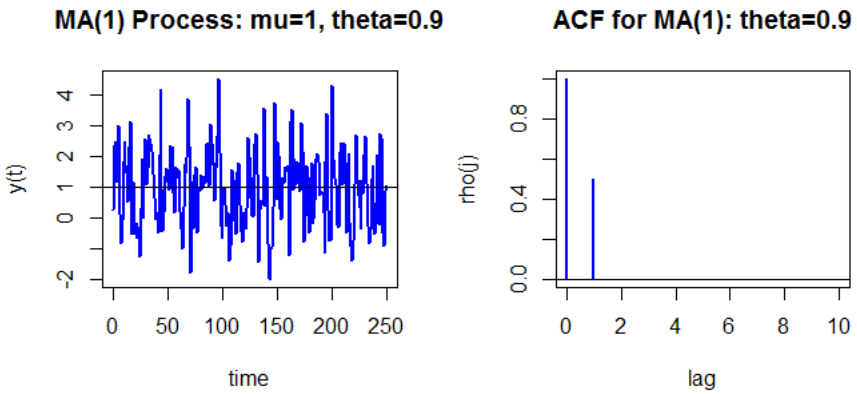


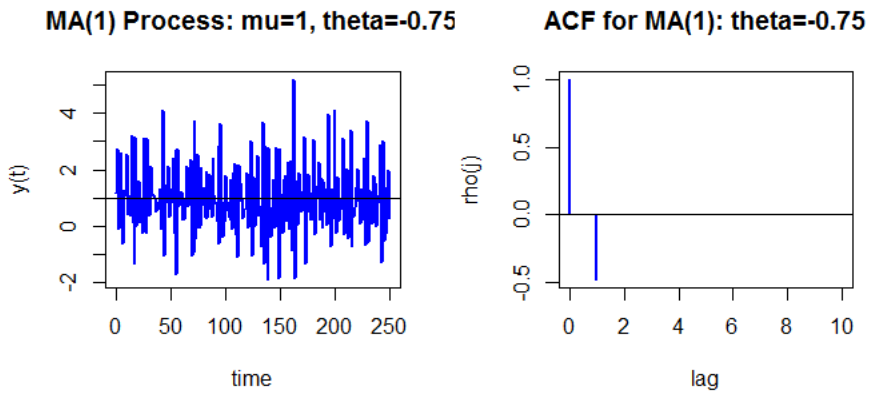
- moving average process

MA(q): $X_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}$

```
# simulate MA(1) process with theta 0.9 and e(t) ~ N(0,1)
```

```r
ma1.model = list(ma=0.9)
mu = 1
set.seed(123)
ma1.sim = mu + arima.sim(model=ma1.model,n=250)

# simulate MA(1) process with theta 0.9 and e(t) ~ N(0,(0.1)^2)
set.seed(123)
ma1.sim2 = mu + arima.sim(model=ma1.model, n=250, innov=rnorm(n=250, mean=0,
sd=0.1))

# ACF for MA(1) model
ma1.acf = ARMAacf(ar=0, ma=0.9, lag.max=10)
ma1.acf

par(mfrow=c(1,2))
ts.plot(ma1.sim,main="MA(1) Process: mu=1, theta=0.9",
        xlab="time",ylab="y(t)", col="blue", lwd=2)
abline(h=1)
plot(0:10, ma1.acf,type="h", col="blue", lwd=2,
     main="ACF for MA(1): theta=0.9",xlab="lag",ylab="rho(j)")
abline(h=0)

# simulate MA(1) process with theta < 0
ma1.model = list(ma=-0.75)
mu = 1
set.seed(123)
ma1.sim = mu + arima.sim(model=ma1.model,n=250)
ts.plot(ma1.sim,main="MA(1) Process: mu=1, theta=-0.75",
        xlab="time",ylab="y(t)", col="blue", lwd=2)
abline(h=1)

# ACF for MA(1) model
ma1.acf = ARMAacf(ar=0, ma=-0.75, lag.max=10)
plot(0:10, ma1.acf,type="h", col="blue", lwd=2,
     main="ACF for MA(1): theta=-0.75",xlab="lag",ylab="rho(j)")
abline(h=0)
```





- Autoregressive process

AR(p):$X_t = c + \sum_{i=1}^{p} \psi_i X_{t-i} + \epsilon_t$

AR(1) examples:

1. Interest rates

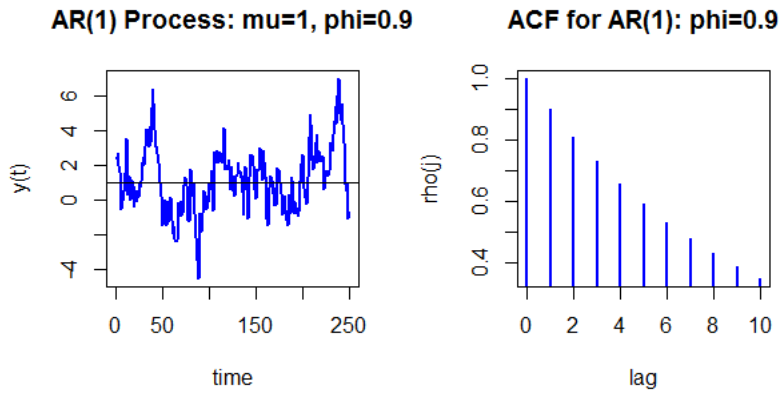
   Ornstein Uhlenbeck Process: $dX_t = -\theta_2 X_t dt + \theta_3 dW_t$

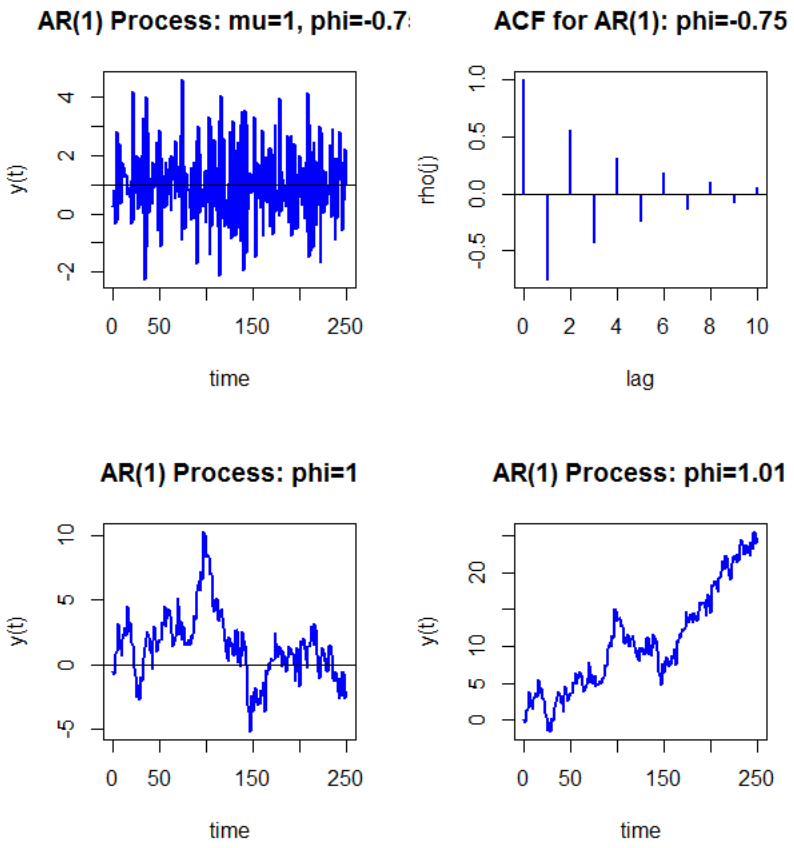
   Vasicek Model: $dX_t = (\theta_1 - \theta_2 X_t)dt + \theta_3 dW_t$
2. Interest rate spreads
3. Real exchange rate
4. Valuation ratios (dividend-to-price, earnings-to-price)

Simulations

```
# simulate AR(1) process: phi = 0.9
ar1.model = list(ar=0.9)
mu = 1
set.seed(123)
ar1.sim = mu + arima.sim(model=ar1.model,n=250)
ar1.acf = ARMAacf(ar=0.9, ma=0, lag.max=10)

par(mfrow=c(1,2))
ts.plot(ar1.sim,main="AR(1) Process: mu=1, phi=0.9",
        xlab="time",ylab="y(t)", col="blue", lwd=2)
abline(h=1)
# ACF for AR(1) model
plot(0:10, ar1.acf,type="h", col="blue", lwd=2,
     main="ACF for AR(1): phi=0.9",xlab="lag",ylab="rho(j)")
par(mfrow=c(1,1))

# simulate AR(1) process: phi = -0.75
ar1.model = list(ar=-0.75)
mu = 1
set.seed(123)
ar1.sim = mu + arima.sim(model=ar1.model,n=250)

# ACF for AR(1) model
```

```r
ar1.acf = ARMAacf(ar=-0.75, ma=0, lag.max=10)

par(mfrow=c(1,2))
ts.plot(ar1.sim,main="AR(1) Process: mu=1, phi=-0.75",col="blue", lwd=2,
        xlab="time",ylab="y(t)")
abline(h=1)
plot(0:10, ar1.acf,type="h", col="blue", lwd=2,
     main="ACF for AR(1): phi=-0.75",xlab="lag",ylab="rho(j)")
abline(h=0)

# simulate AR(1) process: phi = 1
set.seed(123)
ar1.sim = cumsum(rnorm(250))

# simulate AR(1) process: phi > 1
set.seed(123)
phi = 1.01
e = rnorm(250)
y = rep(0,250)
for (i in 2:250) {
      y[i] = phi*y[i-1] + e[i]
}

par(mfrow=c(1,2))
ts.plot(ar1.sim,main="AR(1) Process: phi=1",
        xlab="time",ylab="y(t)",lwd=2, col="blue")
abline(h=0)
ts.plot(y,main="AR(1) Process: phi=1.01",
        xlab="time",ylab="y(t)", lwd=2, col="blue")
par(mfrow=c(1,1))
```





- ARIMA Model Seclection

1. Find d such that $(1 - Lag)^d(Y_t)$ is stationary. Then work with $(1 - Lag)^d(Y_t)$
2. Find (p, q) minimize one of the information criteria under the assumption of Gaussian white noise:
   Akaike Information Criterion

$$AIC = 2k - 2\ln L$$

   Corrected Akaik Information Criterion

$$AIC_C = AIC + \underbrace{\frac{2k(k+1)}{n-k-1}}_{bias-correction}$$

   Akaike's Final Prediction Error

   Bayes Information Criterion or Schwarz information criterion

$$BIC = -2\ln \hat{L} + k\ln n$$

   Hannan-Quinn information Criterionq

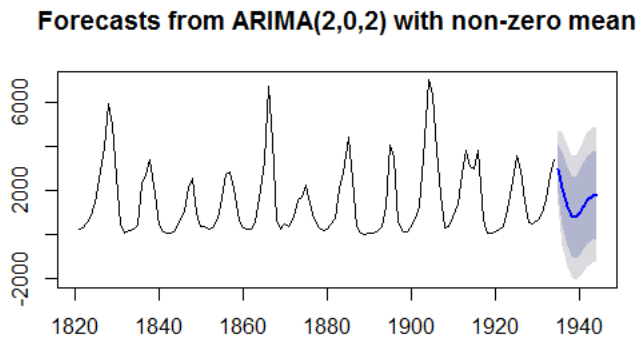
$$HQC = -2L_{max} + 2k\log\log n$$

Where ln L *or* L_{max} is $log(Maximum\ likelihood\ of\ Var(WN))$, n = num of observations, $k = (p+q)/n$

```
library(forecast)

auto.arima(lynx)
auto.arima(lynx, ic = "aic", trace = T)

# ARIMA(2,0,2) with non-zero mean : 1876.391
# ARIMA(0,0,0) with non-zero mean : 2007.339
# ARIMA(1,0,0) with non-zero mean : 1927.298
# ARIMA(0,0,1) with non-zero mean : 1918.958
# ARIMA(0,0,0) with zero mean     : 2081.408
# ARIMA(1,0,2) with non-zero mean : 1888.538
# ARIMA(3,0,2) with non-zero mean : 1878.685
# ARIMA(2,0,1) with non-zero mean : 1879.737
# ARIMA(2,0,3) with non-zero mean : Inf
# ARIMA(1,0,1) with non-zero mean : 1890.909
# ARIMA(3,0,3) with non-zero mean : Inf
# ARIMA(2,0,2) with zero mean     : 1905.759
# Best model: ARIMA(2,0,2) with non-zero mean

# Series: lynx
# ARIMA(2,0,2) with non-zero mean

# Coefficients:
         # ar1      ar2      ma1      ma2  intercept
      # 1.3421  -0.6738  -0.2027  -0.2564  1544.4039
# s.e.  0.0984   0.0801   0.1261   0.1097   131.9242

# sigma^2 estimated as 728546:  log likelihood=-932.08
# AIC=1876.17   AICc=1876.95   BIC=1892.58

# Yt = Int + (ar1)*Yt-1 + (ar2)*Yt-2 + (ma1)*Ye-1 + (ma2)*Ye-2
```

- ARMA forecast

```
myarima = arima(lynx, order = c(2,0,2), include.mean = T) # arima class
object which contains the coeffs

fore<-forecast.Arima(myarima, h = 10) # forecasting next ten values

#   Point    Forecast     Lo 80    Hi 80      Lo 95    Hi 95
# 1935       2989.910 1896.0434 4083.777  1316.9853 4662.835
# 1936       2093.478  435.2392 3751.717  -442.5795 4629.536
# 1937       1307.309 -475.6345 3090.252 -1419.4677 4034.085

plot.forecast(fore)
```

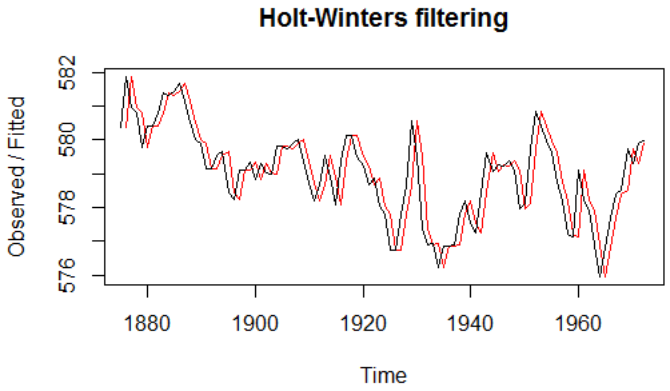


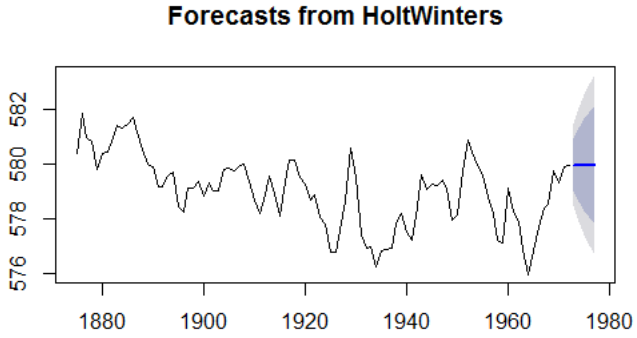
- Exponential Smoothing: Holt Winters

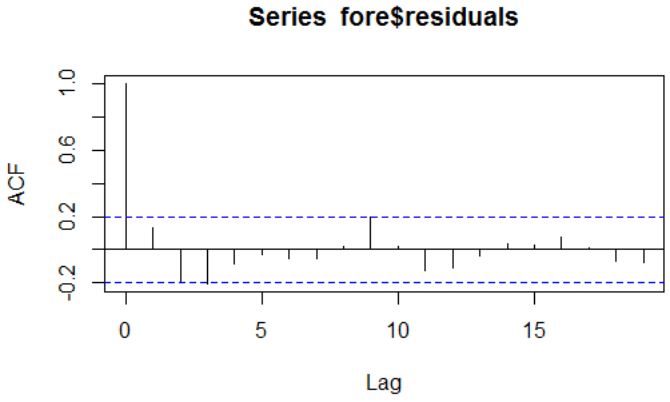
  Examples in python

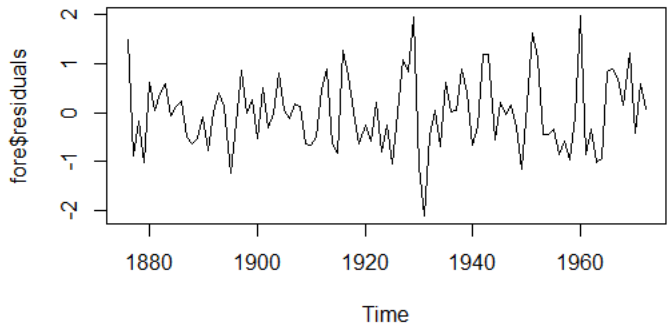
  https://nbviewer.ipython.org/github/thedataincubator/ds30_3/blob/master/ds30_hw_examples.ipynb

```
### non-season, non-trend Example

plot(LakeHuron)
x = HoltWinters(LakeHuron, gamma = F, beta = F);x
plot(x)

library(forecast)
fore <- forecast.HoltWinters(x, h = 5)

plot.forecast(fore)

#three ways to see if residual is uncorrelated, unbiased
acf(fore$residuals) #way 1
plot.ts(fore$residuals) #way 2
Box.test(fore$residuals, lag=20, type="Ljung-Box") #way 3

# Box-Ljung test
# data:  fore$residuals
# X-squared = 25.246, df = 20, p-value = 0.1922
```


Holt-Winters filtering
Observed / Fitted
576
578
580
582
1880
1900
1920
1940
1960
Time


Forecasts from HoltWinters
576
578
580
582
1880
1900
1920
1940
1960
1980


Series fore$residuals
ACF
-0.2
0.2
0.6
1.0
0
5
10
15
Lag



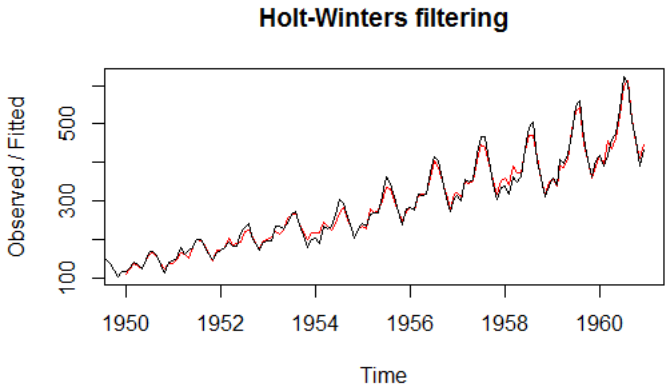
- Exponential Smoothing: Holt Winters with trend (double exponential)

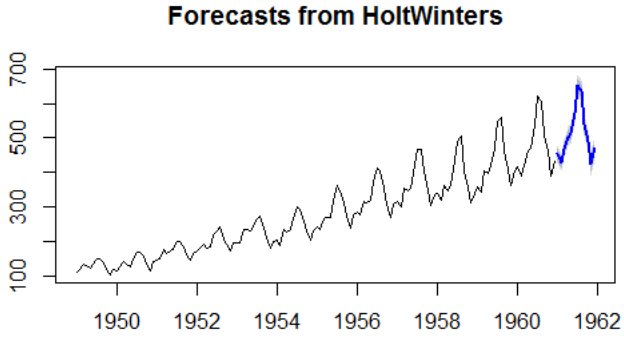
$$s_i = \alpha x_i + (1-\alpha)(s_{i-1} + t_{i-1})$$
$$t_i = \beta(s_i - s_{i-1}) + (1-\beta)t_{i-1}$$

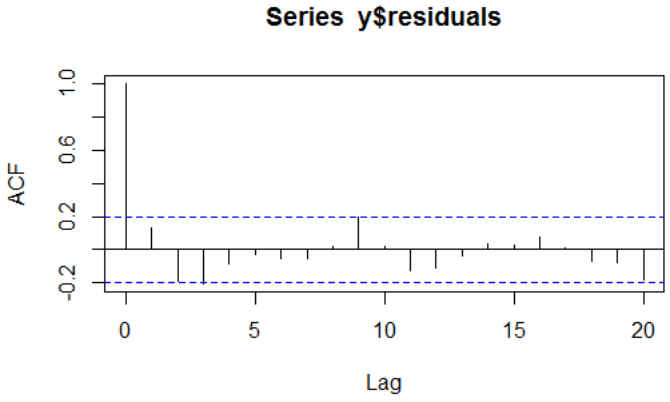
With trend and seasonality (triple exponential)

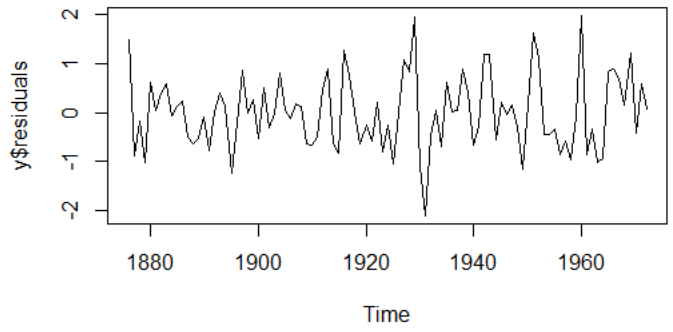
$$s_i = \alpha(x_i - p_{i-k}) + (1-\alpha)(s_{i-1} + t_{i-1})$$
$$t_i = \beta(s_i - s_{i-1}) + (1-\beta)t_{i-1}$$
$$p_i = \gamma(x_i - s_i) + (1-\gamma)p_{i-k}$$

```
# trend, seasonality, maybe additive

plot((AirPassengers))
x<-HoltWinters(AirPassengers)
plot(x)

fore<-forecast.HoltWinters(x, h = 12) # one year forecast
plot.forecast(fore)

acf(y$residuals, lag = 20)
plot.ts(y$residuals)
Box.test(y$residuals, lag=20, type="Ljung-Box") # or log or diff

# Box-Ljung test
# data:  y$residuals
# X-squared = 25.246, df = 20, p-value = 0.1922
```


Holt-Winters filtering
Observed / Fitted
100
300
500
1950
1952
1954
1956
1958
1960
Time


Forecasts from HoltWinters
100
300
500
700
1950
1952
1954
1956
1958
1960
1962

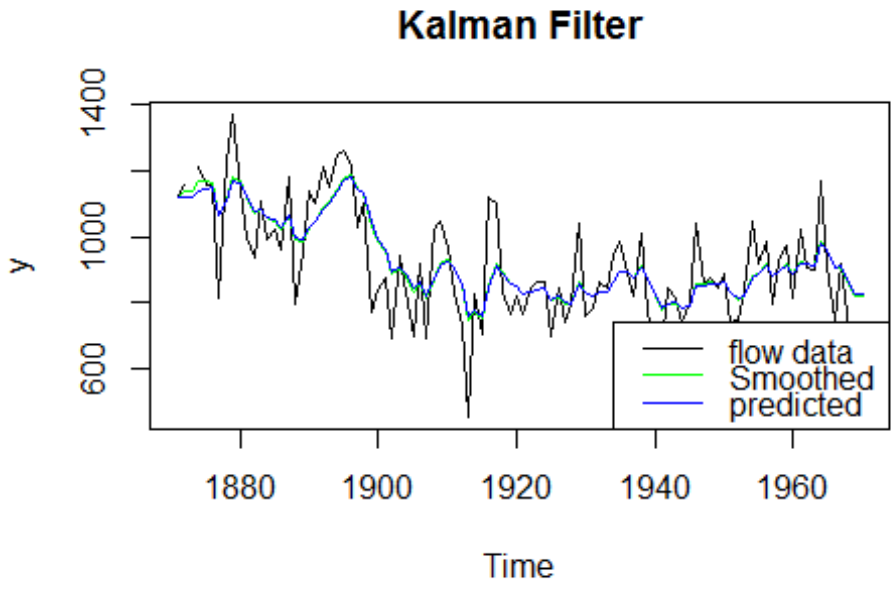

Series y$residuals
ACF
-0.2
0.2
0.6
1.0
0
5
10
15
20
Lag



- Kalman Filter

```
library(FKF)
data("Nile", package = "datasets")
## Transition equation:
## alpha[t+1] = alpha[t] + eta[t], eta[t] ~ N(0, HHt)
## Measurement equation:
## y[t] = alpha[t] + eps[t], eps[t] ~  N(0, GGt)

y <- Nile
y[c(3, 100)] <- NA  # NA values can be handled

## Set constant parameters:
dt <- ct <- matrix(0)
Zt <- Tt <- matrix(1)
a0 <- y[1]            # Estimation of the first year flow
P0 <- matrix(100)     # Variance of 'a0'

## Estimate parameters:
fit.fkf <- optim(c(HHt = var(y, na.rm = TRUE) * .5,
                   GGt = var(y, na.rm = TRUE) * .5),
                 fn = function(par, ...)
                       -fkf(HHt = matrix(par[1]), GGt = matrix(par[2]),
...)$logLik,
                 yt = rbind(y), a0 = a0, P0 = P0, dt = dt, ct = ct,
                 Zt = Zt, Tt = Tt, check.input = FALSE)

## Filter Nile data with estimated parameters:
fkf.obj <- fkf(a0, P0, dt, ct, Tt, Zt, HHt = matrix(fit.fkf$par[1]),
               GGt = matrix(fit.fkf$par[2]), yt = rbind(y))

## Compare with the stats' structural time series implementation:
fit.stats <- StructTS(y, type = "level")

fit.fkf$par
fit.stats$coef

## Plot the flow data together with fitted local levels:
```

```
plot(y, main = "Kalman Filter")
lines(fitted(fit.stats), col = "green")
lines(ts(fkf.obj$att[1, ], start = start(y), frequency = frequency(y)), col =
"blue")
legend("bottomright", c( "flow data", "smoothed", "predicted"),
       col = c("black", "green", "blue"), lty = 1)
```



13. Volatility

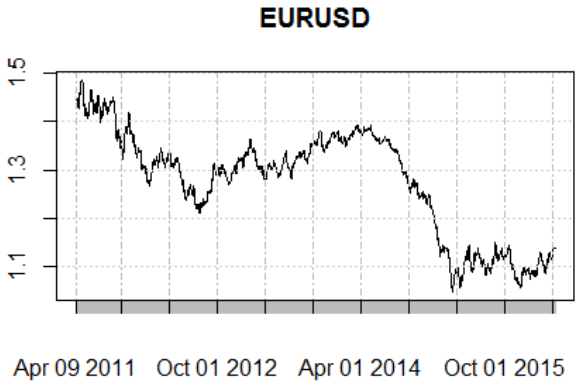
- ARCH
- GARCH

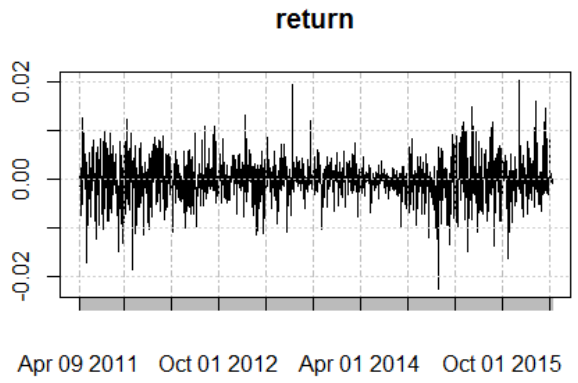
ARCH $\sigma^2 \sim \sum(\theta \cdot \eta_t^2)$ => AR model

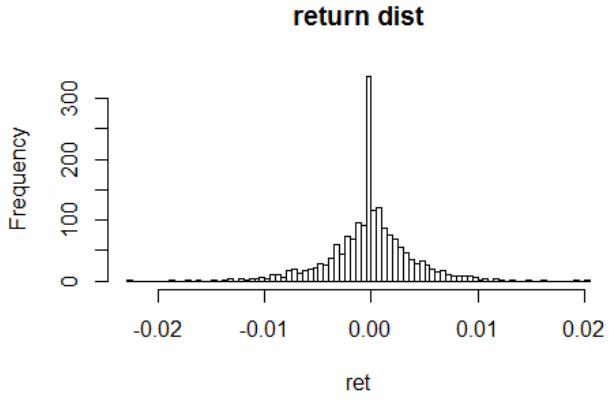
GARCH $\sigma^2 \sim \sum(\theta \cdot \eta_t^2) + \sum(\theta \cdot \sigma_t^2)$ => $ARMA$ model

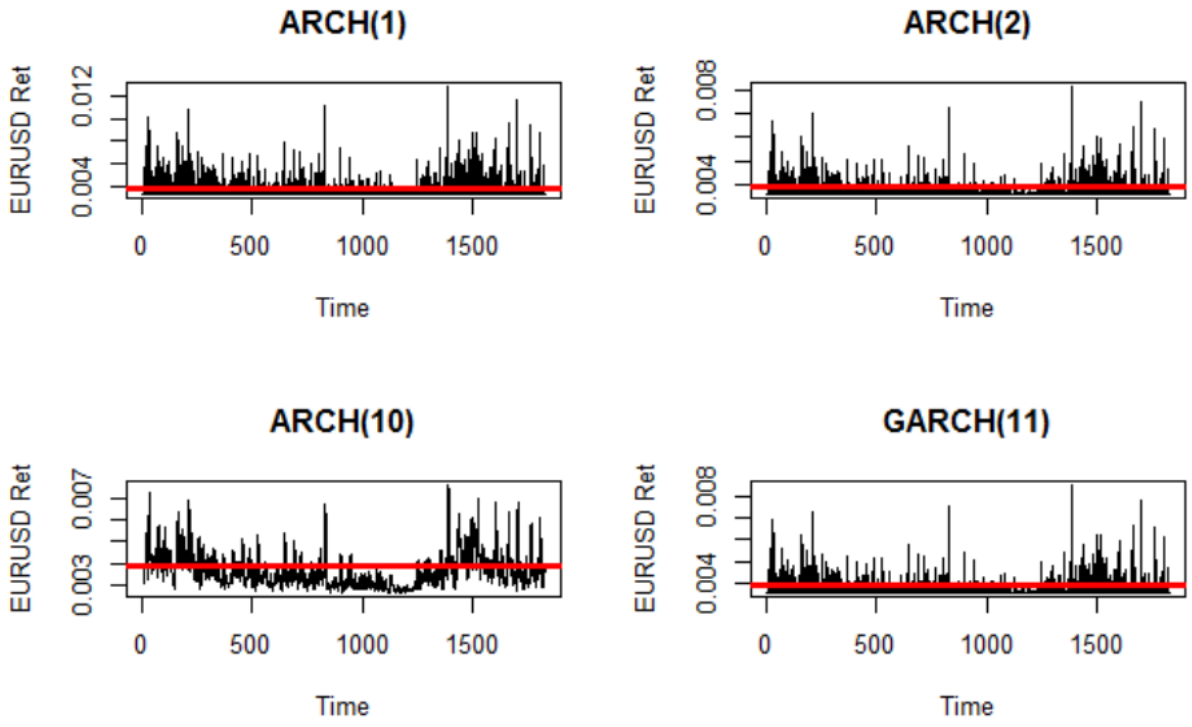
```
library(quantmod)
getFX("EUR/USD",from=as.Date("2011-01-01") )
ret <- dailyReturn(EURUSD, type = "log")
plot(EURUSD)
plot(ret, main="return")
hist(ret, breaks = 100, main = "return dist")

library(tseries)
ret.arch1 <- garch(ret, order = c(0,1), trace=F)
ret.arch2 <- garch(ret, order = c(0,2), trace=F)
ret.arch10 <- garch(ret, order = c(0,10), trace=F)
ret.garch11 <- garch(ret, order = c(1,1), trace=F)

summary(ret.arch1)
#Model:
```

```
#GARCH(0,1)
# Residuals:
#     Min      1Q  Median      3Q     Max
# -6.3794 -0.4905  0.0000  0.4361  5.5797
#
# Coefficient(s):
#     Estimate  Std. Error  t value Pr(>|t|)
# a0 1.103e-05   3.059e-07   36.063   <2e-16 ***
# a1 3.018e-01   3.112e-02    9.699   <2e-16 ***
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#
# Diagnostic Tests:
#    Jarque Bera Test
#
# data:  Residuals
# X-squared = 693.44, df = 2, p-value < 2.2e-16
#
#    Box-Ljung test
#
# data:  Squared.Residuals
# X-squared = 0.15509, df = 1, p-value = 0.6937

par(mfrow=c(2,2))
ts.plot(ret.arch1$fitted.values[,1], ylab="EURUSD Ret"
        , main = "ARCH(1)")
abline(h=sqrt(var(ret)), col=2, lwd=3)

ts.plot(ret.arch2$fitted.values[,1], ylab="EURUSD Ret"
        , main = "ARCH(2)")
abline(h=sqrt(var(ret)), col=2, lwd=3)

ts.plot(ret.arch10$fitted.values[,1], ylab="EURUSD Ret"
        , main = "ARCH(10)")
abline(h=sqrt(var(ret)), col=2, lwd=3)

ts.plot(ret.garch11$fitted.values[,1], ylab="EURUSD Ret"
        , main = "GARCH(11)")
abline(h=sqrt(var(ret)), col=2, lwd=3)
```


EURUSD
1.5
1.3
1.1
Apr 09 2011
Oct 01 2012
Apr 01 2014
Oct 01 2015


return
0.02
0.00
-0.02
Apr 09 2011
Oct 01 2012
Apr 01 2014
Oct 01 2015


return dist
Frequency
0
100
200
300
-0.02
-0.01
0.00
0.01
0.02
ret



Portfolio theory & Risk Management

14. Markowitz algorithm

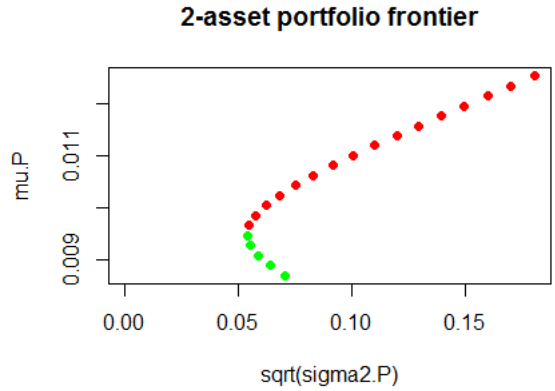
- Portfolio Frontier

Perfect hedge $\rho_p = 0$,

Copied from http://faculty.washington.edu/ezivot/econ424/portfolioTheoryMatrix.r

```
library(tseries)
library(zoo)
end = Sys.Date()
d<- as.POSIXlt(as.Date(end))
d$year <- d$year - 10
start = as.Date(d)
par(mfrow=c(1,1))

MSFT.prices = get.hist.quote(instrument="msft", start=start,
                             end=end, quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")
MSFT = diff(log(MSFT.prices))


SBUX.prices = get.hist.quote(instrument="sbux", start=start,
                             end=end, quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")
SBUX = diff(log(SBUX.prices))
```

```
cb <- cbind(MSFT,SBUX)
mu = colMeans(cb)
sigma2 = apply(cb, 2, var)
rho_ij = cov(cb[,1],cb[,2])

w.MSFT <- seq(-1,1, by = 0.1)
w.SBUX <- 1-w.MSFT

mu.P <- w.MSFT * mu[1] + w.SBUX * mu[2]
sigma2.P <- w.MSFT^2 * sigma2[1] + w.SBUX^2 * sigma2[2]
                  + 2* w.MSFT * w.SBUX * rho_ij
plot(sqrt(sigma2.P), mu.P, type = "b", pch=16,
     col = c(rep("red", 16), rep("green", 5)), main="2-asset portfolio
frontier")
```



- Efficient Portfolio

With a risk free asset, leverage

$$\mu_p = return_f + (sharpe\ ratio) \cdot \sigma_p$$

sharpe ratio = slope of risk return tradeoff → Tangency portfolio + T-bill

add constraints: no short sell $w > 0$, holding threshold $w < w_{max}$, turnover δ_w

optimize techniques: polynomial goal programming (e.g. R- solve.QP {quadprog}), Bellman-Hamilton-Jacobi equation

```
# Markowitz problem 1
# target.return is given, min risk

#for long only portfolio
Dmat <- 2*cov.mat
dvec <- rep.int(0, N)
Amat <- cbind(rep(1,N), er, diag(1,N))
bvec <- c(1, target.return, rep(0,N))
opt_weight <- solve.QP(Dmat=Dmat,dvec=dvec,Amat=Amat,bvec=bvec,meq=2)$solution

#for long/short
ones <- rep(1, N)
top <- cbind(2*cov.mat, er, ones)
```

```
bot <- cbind(rbind(er, ones), matrix(0,2,2))
A <- rbind(top, bot)
b.target <- as.matrix(c(rep(0, N), target.return, 1))
opt_weight <- solve(A, b.target)[1:N]
```

15. Single Index Factor (Sharpe Model) & CAPM

- Beta

$$\beta_i = cov(R_i, R_p) \,/\, var(R_p)$$

then percentage contribution of risk of asset $i = \beta_i \cdot w_i$,
=> CAPM: return is a function of beta
Tobin Thm: every optimal portfolio invests in a combination of risk-free and market portfolio.

- Well Diversification

All non-market variance is diversified away, $var(R_p) = \bar{\beta}^2 \, var(R_M)$

16. Advanced Methods for Risk Analysis & Portfolio Optimization

- Random Matrix Theory

Jim Gatheral, *Random Matrix Theory and Covariance Estimation*
Laurent Laloux, Pierre Cizeau, Marc Potters and Jean-Philippe Bouchaud, *Random Matrix Theory and Finical Correlation*
Marco Avallaneda, *A Holistic (and practical) Approach for Risk Managing Equity Derivatives*

- Dynamic Programming (Merton Model)

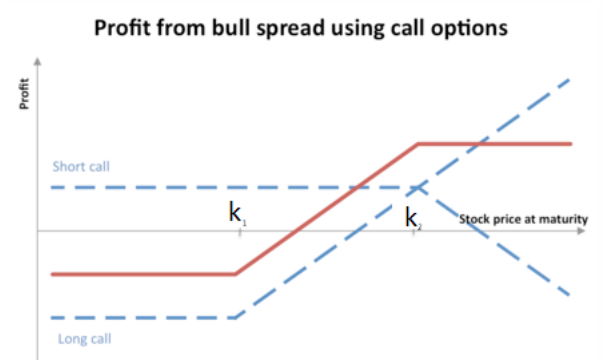
David Brown, James Smith, *Dynamic Portfolio Optimization with Transaction Costs: Heuristics and Dual Bounds*
Yongyang Cai, Kenneth L. Judd, and Rong Xu, *Numerical Solution of Dynamic Portfolio Optimization with Transaction Costs*
Robert Damman, Chester Spatt, and Harold Zhang, *Optimal Consumption and Investment with Capital Gains Taxes*

Derivatives Pricing (Sell Side)

17. Types of derivatives, Binomial Tree, B-S Equation

- Contingent Claims (Hull Ch 8-10)

Forwards $(S_T - K)$, future, calls, puts, European call payoff $C = (S_T - K, 0)_+$, Asian $C = (A(S) - K, 0)_+$, floating strike $C = (S_T - kA(S), 0)_+$,
Bull spread long K_1 call, short K_2, $(S_T - K_1, 0)_+ - (S_T - K_2, 0)_+$



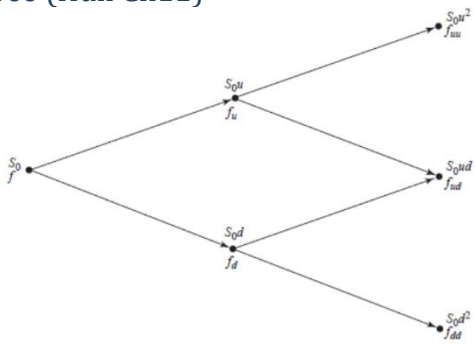
- Interest Derivatives (Hull Ch 4, 7, 26-30)

Bond: LIBOR, Duration. Forward rate.
Swaps & Swaptions.

- Credit Derivatives (Hull Ch 20,21)

Asset swaps, credit-default swaps

- Binomial, trinomial Tree (Hull Ch11)



Two-step Bi-tree

$$f = e^{-2r\Delta t}[p^2 f_{uu} + 2p(1-p) + (1-p)^2 f_{dd}]$$

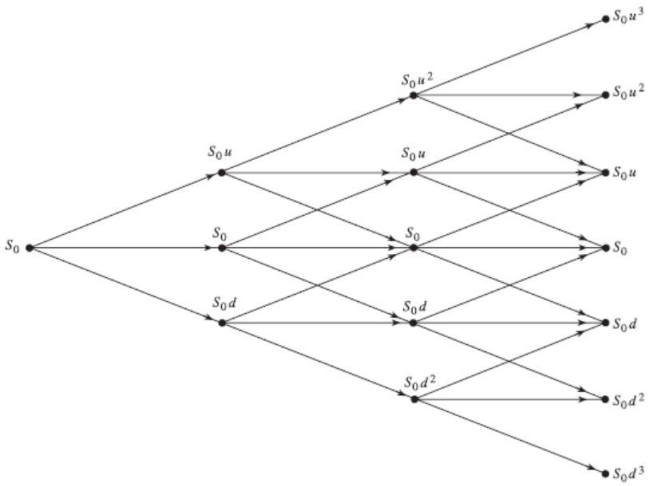
$$p = \frac{e^{r\Delta t} - d}{u - d}$$

p independent of the probabilities of S going up or down

For N step, C=

$$C = e^{-rN\delta t}\sum_{j}^{N}\left[\binom{N}{j} q^j (1-q)^{N-j}\left(s_0 u^j d^{N-j} - K\right)_+\right]$$

Let $N \rightarrow$ inf, assume lognormal, $S \Rightarrow BS$



Trinomial Tree

- Stochastic Process

$$dX_t = a(X_t)dt + b(X_t)dW_t$$

Example (Heston Stochastic volatility model):

$$\frac{dS}{S} = rdt + \sqrt{V}dW$$
$$dV = a(b - V)dt + \sigma\sqrt{V}dW$$

Another risk models: Dupire – local volatility

Numerical methods:

Euler–Maruyama method

$$Y_{n+1} = Y_n + a(Y_n)\Delta t + b(Y_n)\Delta W_n$$

$\Delta W_n = W_{\tau_n+1} - W_{\tau_n}$ are iid $N(0, \Delta t)$

Milstein approximation

$$Y_{n+1} = Y_n + a(Y_n)\Delta t + b(Y_n)\Delta W_n + \frac{1}{2}b(Y_n)b'^{(Y_n)}((\Delta W_n)^2 - \Delta t)$$

Runge–Kutta approximation

$$Y_{n+1} = Y_n + a(Y_n)\Delta t + b(Y_n)\Delta W_n + \frac{1}{2}\big(b(\gamma_n) - b(Y_n)\big)((\Delta W_n)^2 - \Delta t)\sqrt{\Delta t}$$

Where $\Delta t = T/N$, $\gamma_n = Y_n + a(Y_n)\Delta t + b(Y_n)\sqrt{\Delta t}$

Girsanov Theorem:
For any function $\gamma(s)$

$$E\left[e^{\int_a^b \gamma(s)dw - \frac{1}{2}\int_a^b \gamma^2(s)ds}\right] = 1$$

Hence the Radon-Nikodyn derivative on the path space has probability measures.

- Geometric random walk, Ito lemma (Hull Ch 12)

$$\frac{dS}{S} = \mu dt + \sigma dw$$

dw = wiener process. S(t) = stock price. Solution

$$S_t = S_0 e^{\left(\mu + \frac{\sigma^2}{2}\right)t + \sigma W_t}$$

- Black–Scholes (Hull Ch13)

Apply Ito to option value $f(S,t)$

$$df(S,t) = \left(\frac{\partial f}{\partial S}\mu(S,t)S + \frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}(\sigma S)^2\right)dt + \frac{\partial f}{\partial S}(\sigma S)dw$$

Let Π be value of portfolio of one long option f and one short position S in some quantity Δ, i.e.

$$\Pi = f - \Delta S$$

Or

$$d\Pi = df - \Delta dS$$

With some manipulation, we arrive

$$d\Pi = \left(\frac{\partial f}{\partial S} - \Delta\right)dS + \left(\frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}(\sigma S)^2\right)dt$$

The first part dS contains Wiener random term, so we will choose $\Delta = \partial f/\partial S$, called Δ hedging, an example of dynamic hedging strategy, then

$$d\Pi = \left(\frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}(\sigma S)^2\right)dt$$

Set $d\Pi = r\Pi dt$, we get the BS equation

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial S}rS + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}(\sigma S)^2 = rf$$

2nd order pde need to 2 boundary conditions, see later section on finite-difference method

- Ornstein-Uhlenbeck Process – Vasicek interest rate model—mean reversion

$$dr = a(b - r)dt + \sigma dw$$

Solution

$$r = b + e^{-at}(r_0 + b) + \sigma \int_0^t e^{a(s-t)} dw(s)$$

With

Mean = $b + e^{-at}(r_0 + b) \to b$.Var = $\frac{\sigma^2(1-e^{-2at})}{2a} \to \frac{\sigma^2}{2a}$

- Cox–Ingersoll–Ross (CIR) model—short rate model—mean reversion – positive rate

$$dr_t = a(b - r_t)dt + \sigma\sqrt{r_t} dW_t$$

Mean = $b + e^{-at}(r_0 - b) \to b$.Var → $\frac{b\sigma^2}{2a}$

- Jump diffusion Process

$$\frac{dS}{S} = \mu dt + \sigma dW + (q - 1)dJ$$

J is a Poisson process. Time between successive jumps are independent and exponentially distributed with parameter λ , $E[J_t] = var(J_t) = \lambda t$.

B-S equation becomes partial integro-differential equation

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - \lambda c)S\frac{\partial V}{\partial S} - (\lambda + r)V + \lambda \mathbb{E}\big(V(qS, t)\big) = 0$$

18. Implied Volatility, Greeks (Hull Ch 15-16, 19)

Given option price, using BS to find => implied volatility

Greeks

$$\Delta = \frac{\partial f}{\partial S_0} \text{ (price movement)}$$

$$\gamma = \frac{\partial^2 f}{\partial S_0^2} \text{ (price movement)}$$

$$\theta = \frac{\partial f}{\partial(-T)} \text{ (time decay)}$$

$$\text{Vega} = \frac{\partial f}{\partial \sigma} \text{ (implied volatility)}$$

$$\rho = \frac{\partial f}{\partial r}$$

S_0= stock price. For options on a forward rate, change S_0 to F_0

19. European option - vanilla (Hull Ch 14)

Call

When $t = T$

$$c = \max(S_T - K, 0)$$

or

$$c(t) = e^{-r(T-t)}\mathbb{E}[\max(S_T - K, 0)]$$

Using a key result (Hull, page 310)

$$\mathbb{E}[\max(S - K, 0)] = \mathbb{E}[S_T]N(d_1) - KN(d_2)$$

Assume S_T is lognormal wrt to $t = 0$,

$$c(t = 0) = S_0 N(d_1) - Ke^{-rT}N(d_2)$$

Where $N = \frac{\int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt}{\sqrt{2\pi}}$

$$d_1 = \frac{\ln\frac{S_0}{K} + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} \quad d_2 = \frac{\ln\frac{S_0}{K} + \left(r - \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

Monte Carlo

$$C = E\left[e^{-rT}\left(S_0 e^{\left(r-\frac{\sigma^2}{2}\right)T+\sigma\sqrt{T}Z} - K\right)_+\right], \qquad Z \sim N(0,1)$$

20. Exotic - path dependent

- Asian option

No dividend -> never exercise early (if deep in money, sell the options instead)

$$C = (AS_T - K, 0)_+$$

Average S can be continuous or discrete, arithmetic or geometric.

$$AS_{AC} = \frac{1}{T}\int_0^T S(t)dt \qquad AS_{AD} = \frac{1}{N}\sum_{i=1}^{N} S(t_i)$$

$$AS_{GC} = e^{\left(\frac{1}{T}\int_0^T \ln(S(t))dt\right)} \quad AS_{AD} = \left(\prod_{i=1}^{N} S(t_i)\right)^{\frac{1}{N}}$$

Geometric Asian AS_{GC} has a known analytic solution

$$C_{GC} = V_0 N(d_1) - e^{-rT}KN(d_2)$$

$$d_1 = \frac{1}{\sigma_{ave}\sqrt{T}}\log\left(\frac{V_0}{K} + \left(r + \frac{\sigma_{ave}^2}{2}\right)T\right), \quad d_2 = d_1 - \sigma_{ave}\sqrt{T}, \qquad \sigma_{ave} = \frac{\sigma}{\sqrt{3}} \;\; \sigma_{ave}$$

$$V_0 = e^{-\frac{1}{12}(6r+6q+\sigma^2)T}S_0$$

q = dividend yield

Monte Carlo

$$C = E\left[e^{-rT}\left(\frac{1}{m}\sum_{i=1}^{m} S_{\frac{iT}{m}} - K\right)_+\right]$$

$$S_{\frac{iT}{m}} = S_0 e^{\left(r-\frac{\sigma^2}{2}\right)\frac{iT}{m}+\sigma\sqrt{\frac{T}{m}}(Z_1+\cdots+Z_i)}, \qquad Z_{1,\ldots,i} \sim N(0,1)$$

- Double Barrier Option

$$C = \begin{cases} (S_T - K_1)^+ & S_{T/2} \le L \\ (S_T - K_2)^+ & \text{othewise} \end{cases}$$

- Lookback Option

$$C = (S_{max} - K)^+$$

21. Variance Reduction Techniques

- Antithetic Variates

Use *n* 2-pair of paths

$$S(t_i + \Delta t)_\pm = S(t_i)(1 + r\Delta t \pm \sigma\sqrt{\Delta t} Z_{i+1})$$

To get S_T, then the estimator $Y = e^{-rT}\left(S_{T_\pm} - K, 0\right)_+$ for European option. The standard deviation of Y from the 2n paths (related to confidence interval) is

$$\frac{\text{std}\left(\frac{Y_{T_+} + Y_{T_-}}{2}\right)}{\sqrt{n}}$$

- Control Variates

We need to estimate Y and to reduce Var(Y). Instead of estimating Y, we estimate

$$\theta = Y - c(Z - E(Z))$$

because $E[Y] = E[\theta]$. To make Var(θ) small, we choose

$$c = \frac{\text{Cov}(Y, Z)}{\text{Var}(Z)}$$

Hence the slope of linear regression of (Z, Y). The higher Corr(Y, Z), the more reduction it gets

Step 0. Pick a Z. Z can be S_T (most reduction), AS_{AD}
Step 1. Pilot simulation. Generate 100 (Y, Z). Compute c

Step 2. Main simulation. Generate 900 (Y, Z). Use above c, get 900 θ. Then report $E[\theta]$, Var$[\theta]$

- Stratified Sampling
  - Proportion Stratification

Let $\mathcal{I}$ be some non-overlapping partition of Y. One can get variance reduction, instead of estimating

$$E[Y]$$

By estimating

$$\sum_{I\in\mathcal{I}} E[Y|I]E[I] = E[Y]$$

and treating $E[I]$ from analytically known value, then estimating $Var(Y)$ becomes estimating

$$\sum_{I\in\mathcal{I}} Var(E[Y|I])E[I]$$

and we know it's $< Var(Y)$, because of the identity

$$Var(Y) = \sum_{I\in\mathcal{I}} Var(E[Y|I])E[I] + E[Var(Y|I)]$$

  - Terminal Stratification & Brownian Bridge

Up-and-out option, say we first generate the mid points of the paths, $W_{T/2}\sim N(0, T/2)$. If that results $S_{T/2} > L$, we can forget the entire path. That is after generating useful W_T, we go backward to generate $(W_h|W_0, W_T)$, $(W_{2h}|W_h, W_T)$, $\ldots, (W_{T-h}|W_{T-2h}, W_T)$, using the following Brownian bridge formula

$$(W_t|W_u = x, W_v = y)\sim N\left(\frac{(v-t)x + (t-u)y}{v-u}, \frac{(v-t)(u-t)}{v-u}\right) \text{ for } u < t < v$$

- Importance Sampling

Monte Carlo says to evaluate $E_f[h(X)]$, first generate a bunch of X_i satisfying the probability density function $f(x)$, then compute

$$mean\big(h(X_i)\big) \approx E_f[h(X)]$$

By the following identity

$$E_f[h(X)] = \int h(x)f(x)dx = \int h(x)\frac{f(x)}{g(x)}g(x)dx = E_g[\frac{h(X)f(X)}{g(X)}]$$

One can equivalently generate a bunch of X_i accord to the probability density function g, then compute

$$mean\left(\frac{h(X_i)f(X_i)}{g(X_i)}\right) \approx E_f[h(X)]$$

To achieve a variance reduction,

$$Var_f\big(h(X_i)\big) - Var_g\left(\frac{h(X_i)f(X_i)}{g(X_i)}\right) = \int h^2(x)\left(1 - \frac{f(x)}{g(x)}\right)f(x)dx > 0$$

Since f, g are pdf, we cannot force $f < g\ \forall x$. We want to pick a g such that $f < g$ on the support where hf are large; $f > g$ when hf are small.

22. Quasi-Monte Carlo Method

- Low Discrepancy Sequences

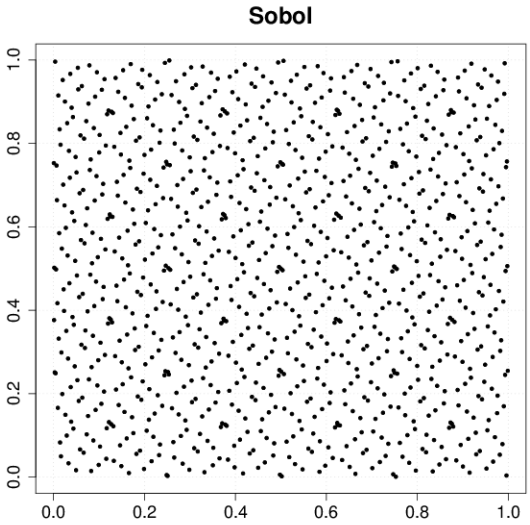
Let $\mathcal{A}$ be collection of rectangles of the form

$$\prod_{j=1}^{d}[u_j, v_j), \qquad 0 \le u_j < v_j \le 1$$

The discrepancy of a sequence $D(x_1, \dots, x_n: \mathcal{A})$ relative to $\mathcal{A}$ is

$$D(x_1, x_2, \dots, x_n: A) = \sup_{A\in\mathcal{A}}\left|\frac{\#\{x_i \in A\}}{n} - vol(A)\right|$$

Implementation: Sobol sequence
Stephen Joe, Frances Y. Kuo, *Notes on generating Sobol sequences*
Source code by John Burkardt: C++(test),
by Corrado Chisari, Python(test)



Picture from http://doc.openturns.org/openturns-latest/html/UseCasesGuide/cid3.xhtml

23. America & Bermudian Options

The formulas, in this section, are copied from Paul Wilmott on *Quantitative Finance 3 Volume Set, section 78*

Write BS as

$$\frac{\partial V}{\partial t} + \underbrace{\frac{1}{2}\sigma^2 S^2}_{a(S,t)} \frac{\partial^2 V}{\partial S^2} + \underbrace{rS}_{b(S,t)} \frac{\partial V}{\partial S} + \underbrace{(-r)}_{c(S,t)} V = 0$$

- Explicit Finite Difference Method (forward in time)

Time derivative is right derivative,

$$\frac{V_i^k - V_i^{k+1}}{\delta t} + a_i^k \left(\frac{V_{i+1}^k - 2V_i^k + V_{i-1}^k}{\delta S^2} \right) + b_i^k \left(\frac{V_{i+1}^k - V_{i-1}^k}{2\delta S} \right) + c_i^k V_i^k = O(\delta t, \delta S^2)$$

k decreases as time goes, i in stock prices, then

$$V_i^{k+1} = A_i^k V_{i-1}^k + \left(1 + B_i^k\right) V_i^k + C_i^k V_{i+1}^k$$
$$A_i^k = v_1 a_i^k - \frac{1}{2} v_2 b_i^k, \qquad B_i^k = -2v_1 a_i^k + \delta t c_i^k$$
$$C_i^k = v_1 a_i^k + \frac{1}{2} v_2 b_i^k$$
$$v_1 = \frac{\delta t}{\delta S^2}, \qquad v_1 = \frac{\delta t}{\delta S}$$

With 2 boundary conditions: $\forall k, S = 0, V = 0; S \to \infty, V = S - Ke^{-r(T-t)}$
With initial condition: $V =$ payoff at $k = T$ for given S_T. If S_T not on the grid, uses linear interpolation.

The solution is stable iff

$$\delta t \leq \frac{\delta S^2}{2a} = \frac{1}{\sigma^2} \left(\frac{\delta S}{S} \right)^2$$

- Implicit Finite Difference Method

Now we let k increase as time goes, i.e. the time derivative is left derivative, which is amount to switch k←→k+1 above

- Crank–Nicolson Finite Difference Method

Take average of (im + explicit FD) => Crank–Nicolson

$$-A_i^{k+1} V_{i-1}^{k+1} + \left(1 - B_i^{k+1}\right) V_i^{k+1} - C_i^{k+1} V_{i+1}^{k+1} = A_i^k V_{i-1}^k + \left(1 + B_i^k\right) V_i^k + C_i^k V_{i+1}^k$$

Where

$$A_i^k = \frac{1}{2}v_1 a_i^k - \frac{1}{4}v_2 b_i^k = \frac{A_{i\ before}^k}{2}, \qquad B_i^k = -v_1 a_i^k + \frac{1}{2}\delta t c_i^k = \frac{B_{i\ before}^k}{2}$$
$$C_i^k = \frac{1}{2}v_1 a_i^k + \frac{1}{4}v_2 b_i^k = \frac{C_{i\ before}^k}{2}$$

And stability holds for all $\Delta t > 0$.

- Successive Over-Relaxation (SOR)

Suppose we solve a matrix equation for *v*

$$Mv = 1$$

Gauss-Seidel iterative method

$$v_i^{n+1} = \frac{1}{M_{ii}}\left(q_i - \sum_{j=1}^{i-1} M_{ij}v_j^{n+1} - \sum_{j=i+1}^{N} M_{ij}v_j^n\right)$$

with some initial guess v^0. SOR method improve convergence speed with over-relaxation parameter $\omega \in [1,2]$

$$v_i^{n+1} = v_i^n + \frac{\omega}{M_{ii}}\left(q_i - \sum_{j=1}^{i-1} M_{ij}v_j^{n+1} - \sum_{j=1}^{N} M_{ij}v_j^n\right)$$

Try adjusting ω, to get the best convergence.

Projected SOR for early exercise

$$v_i^{n+1} = \max\left(v_i^n + \frac{\omega}{M_{ii}}\left(q_i - \sum_{j=1}^{i-1} M_{ij}v_j^{n+1} - \sum_{j=1}^{N} M_{ij}v_j^n\right), \text{payoff}\right)$$

- Parallel LU algorithms of computing band matrices (tridiagonal matrix)

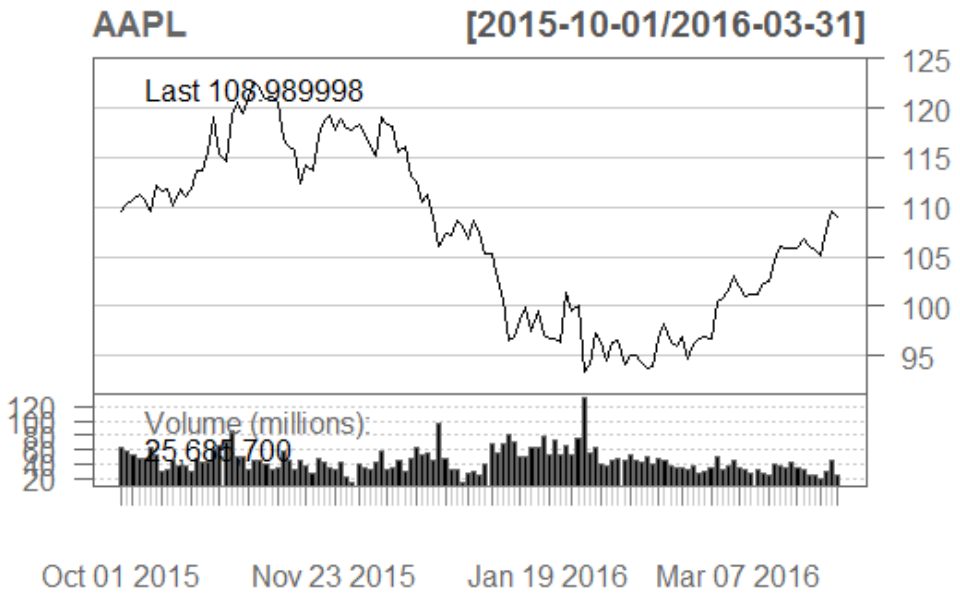
See My Numerical Analysis Note

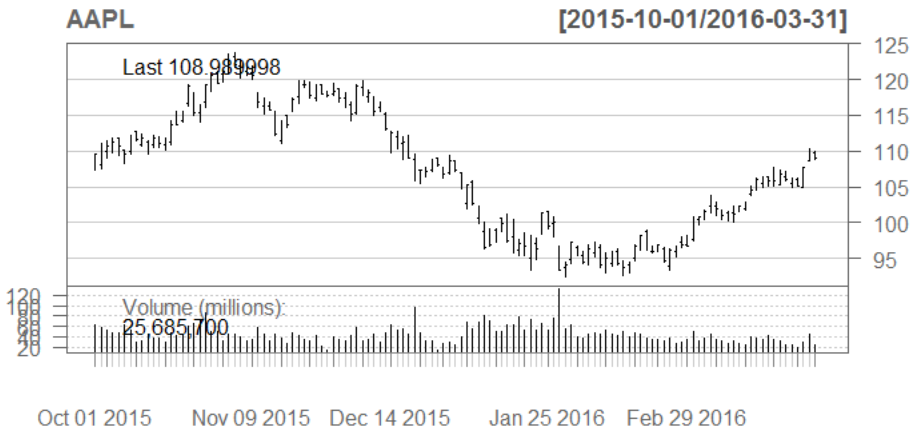
(Algorithmic) Stock/ETF Option Trading (Buy Side)

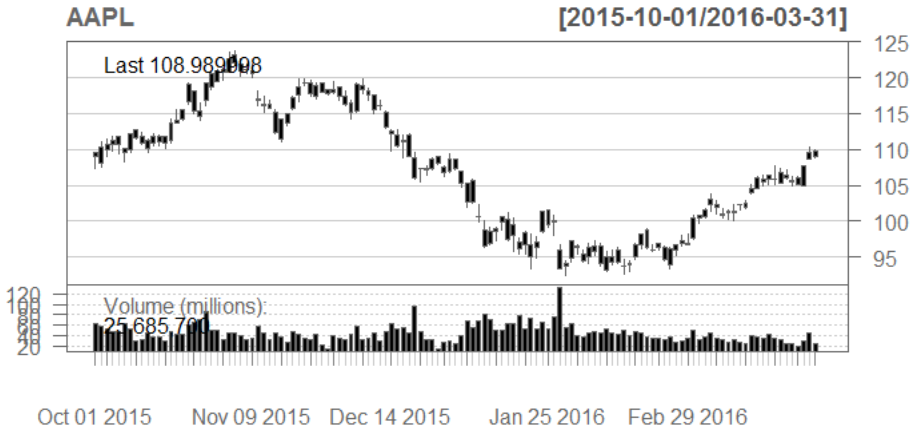
24. Chat Readings

- Dow theory
  volume dictates trend: higher high / lower low / sideways, three phases, trend break
- Chat Patterns
  head, shoulders, double / triple top, double bottom, rising / filling wedges, symmetrical / right angled / descending triangle, pennants, flags, supports, resistance, rectangle, diamond
- Bar Pattern
  two bar reversal, outside / inside bar, exhaustion bar, gaps: breakaway / runaway / exhaustion / common

```
library(quantmod)
getSymbols("AAPL", src="yahoo")

lineChart(AAPL,theme = "white.mono", subset = "last 6 month")

barChart(AAPL,theme='white.mono', subset = "last 6 month")

candleChart(AAPL,theme='white.mono', subset = "last 6 month")

#price in log
lineChart(AAPL,theme = "white", log.scale = T)
```

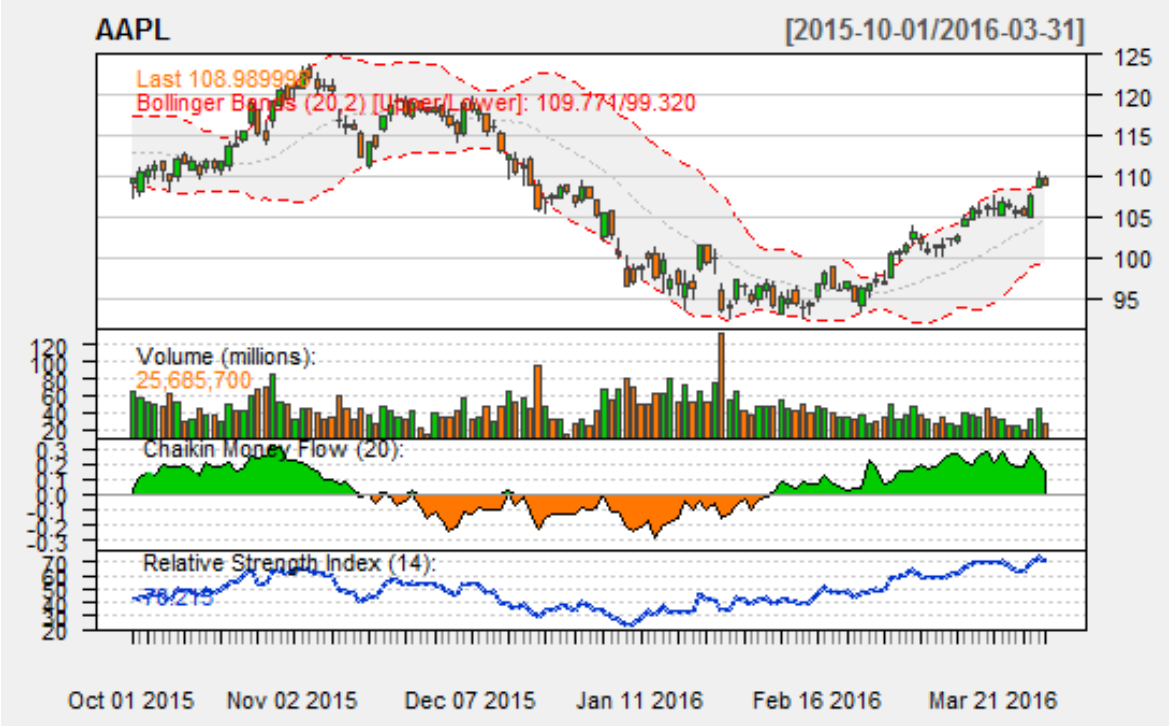






- Leading Indicators: relative strength(addRSI), stochastic oscillator(addSMI, addCMO, addDPO, addTRIX), commodity channel Index(addCCI), Williams %R(addWPR)
- Lagging Indicators: moving average(addDEMA, addEMA, addEVWMA, addSMA) , Bollinger Bands(addBBands),Welles Wilder's (addADX) , Parabolic (addSAR), Moving Average Convergence Divergence(addMACD)
- 4 types of indicators
  oscillators (centered: rate of change, CCI, banded: RSI, Stochastic), trend(lagging)/momentum (MACD), volatility (Bollinger, Chaikin), volume (On-Balance Volume)

```
library(quantmod)
getSymbols("AAPL", src="yahoo")

chartSeries(AAPL,theme="white",
            TA=c(addVo(),addCMF(),addBBands(),addRSI()), subset='last 6
months')
```



- Elliott wave(fibonacci), Daylight (use Golden Ratio), Neural Network Trading Strategy

See Machine Learning Note – Neural Network Algorithms

- News Sentiment

See Machine Learning Note – Natural Language Processing

25. On-line trading: Quantopian open, interactive Brokers, TDAmeritrade, MetaTrader, Forex

26. Common Trading Algorithms

- Based on sectional PE Fundamental Data

Code modified from https://www.quantopian.com/help

```
"""
1. Filter the top 100 companies by market cap
2. Find the top two sectors that have the highest average PE ratio
3. Every month clear the book before entering new ones
"""

import numpy as np

def initialize(context):
    # Dictionary of stocks and their respective weights
    context.stock_weights = {}
    # Count of days before rebalancing
```

```
    context.days = 0
    # Number of sectors to go long in
    context.sect_numb = 2

    # Sector mappings
    context.sector_mappings = {
       101.0: "Basic Materials",
       102.0: "Consumer Cyclical",
       103.0: "Financial Services",
       104.0: "Real Estate",
       205.0: "Consumer Defensive",
       206.0: "Healthcare",
       207.0: "Utilities",
       308.0: "Communication Services",
       309.0: "Energy",
       310.0: "Industrials",
       311.0: "Technology"
    }

    # Rebalance monthly on the first day of the month at market open
    schedule_function(rebalance,
                      date_rule=date_rules.month_start(),
                      time_rule=time_rules.market_open())

    schedule_function(record_positions,
                      date_rule=date_rules.month_start(),
                      time_rule=time_rules.market_close())

"""
Called before the start of each trading day.
"""
def before_trading_start(context, data):

    num_stocks = 100

    # Setup SQLAlchemy query to screen stocks based on PE ratio
    # and industry sector. Then filter results based on market cap
    # and shares outstanding. We limit the number of results to
    # num_stocks and return the data in descending order.
    fundamental_df = get_fundamentals(
        query(
            fundamentals.valuation_ratios.pe_ratio,
            fundamentals.asset_classification.morningstar_sector_code
        )
        .filter(fundamentals.valuation.market_cap != None)
        .filter(fundamentals.valuation.shares_outstanding != None)
        .order_by(fundamentals.valuation.market_cap.desc())
        .limit(num_stocks)
    )

    # Find sectors with the highest average PE
    sector_pe_dict = {}
    for stock in fundamental_df:
        sector = fundamental_df[stock]['morningstar_sector_code']
        pe = fundamental_df[stock]['pe_ratio']

        # If it exists add our pe to the existing list.
        # Otherwise don't add it.
```

```
        if sector not in sector_pe_dict:
          sector_pe_dict[sector] = []

        sector_pe_dict[sector].append(pe)

    # Find average PE per sector
    sector_pe_dict = dict([
        (sectors, np.average(sector_pe_dict[sectors]))
        for sectors in sector_pe_dict
        if len(sector_pe_dict[sectors]) > 0
    ])

    # Sort in ascending order
    sectors = sorted(
                  sector_pe_dict,
                  key=lambda x: sector_pe_dict[x],
                  reverse=True
              )[:context.sect_numb]

    # Filter out only stocks with that particular sector
    context.stocks = [
        stock for stock in fundamental_df
        if fundamental_df[stock]['morningstar_sector_code'] in sectors
    ]

    # Initialize a context.sectors variable
    context.sectors = [context.sector_mappings[sect] for sect in sectors]

    # Update context.fundamental_df with the securities (and pe_ratio)
    # that we need
    context.fundamental_df = fundamental_df[context.stocks]


def create_weights(stocks):
    """
        Takes in a list of securities and weights them all equally
    """
    if len(stocks) == 0:
        return 0
    else:
        weight = 1.0 / len(stocks)
        return weight

def rebalance(context, data):
    # Exit all positions before starting new ones
    for stock in context.portfolio.positions:
        if stock not in context.fundamental_df and data.can_trade(stock):
            order_target_percent(stock, 0)

        # Create weights for each stock
    weight = create_weights(context.stocks)

    # Rebalance all stocks to target weights
    for stock in context.fundamental_df:
        if data.can_trade(stock):
          if weight != 0:
              code = context.sector_mappings[
                   context.fundamental_df[stock]['morningstar_sector_code']
```

```
            ]
        order_target_percent(stock, weight)

def handle_data(context, data):
    """
        Call every minute by the Quantopain simulation engine
    """
    pass
```

- Mean reversion

Code copied from https://www.quantopian.com/help

```
"""
It orders stocks from the top 1% of the previous day's dollar-volume (liquid
stocks).

Hypotheses:
Top-performing stocks from last week will do worse this week, and vice-versa.

On each Monday rank high dollar-volume stocks based on their previous week

Long the bottom 10% of stocks with the WORST returns
Short the top 10% of stocks with the BEST returns

This algorithm can be used in live trading, in the Quantopian Open.
"""

from quantopian.algorithm import attach_pipeline, pipeline_output
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.factors import AverageDollarVolume, Returns


def initialize(context):
    context.long_leverage = 0.5
    context.short_leverage = -0.5
    context.returns_lookback = 5

    # Rebalance on the first trading day of each week at 11AM.
    schedule_function(rebalance,
                      date_rules.week_start(days_offset=0),
                      time_rules.market_open(hours=1, minutes=30))

    # Record tracking variables at the end of each day.
    schedule_function(record_vars,
                      date_rules.every_day(),
                      time_rules.market_close(minutes=1))

    # Create and attach our pipeline (dynamic stock selector), defined below.
    attach_pipeline(make_pipeline(context), 'mean_reversion_example')


def make_pipeline(context):

    pipe = Pipeline()

    # Create a dollar_volume factor using default inputs and window_length.
```

```
    # This is a builtin factor.
    dollar_volume = AverageDollarVolume(window_length=1)
    pipe.add(dollar_volume, 'dollar_volume')

    # Create a recent_returns factor with a 5-day returns lookback. This is
    # a custom factor defined below (see RecentReturns class).
    recent_returns = Returns(window_length=context.returns_lookback)
    pipe.add(recent_returns, 'recent_returns')

    # Define high dollar-volume filter to be the top 5% of stocks by dollar volume.
    high_dollar_volume = dollar_volume.percentile_between(95, 100)

    # Define high and low returns filters to be the bottom 10% and top 10% of
    # securities in the high dollar-volume group.
    low_returns = recent_returns.percentile_between(0,10,mask=high_dollar_volume)
    high_returns =
recent_returns.percentile_between(90,100,mask=high_dollar_volume)

    # Factors return a scalar value for each security in the entire universe
    # of securities.
    pipe.add(recent_returns.rank(mask=high_dollar_volume), 'recent_returns_rank')

    # Add a filter to the pipeline such that only high-return and low-return
    # securities are kept.
    pipe.set_screen(low_returns | high_returns)

    # Add the low_returns and high_returns filters as columns to the pipeline
    pipe.add(low_returns, 'low_returns')
    pipe.add(high_returns, 'high_returns')

    return pipe


def before_trading_start(context, data):
    """
    Called every day before market open.
    """

    # Pipeline_output
    context.output = pipeline_output('mean_reversion_example')

    # Sets the list of securities we want to long as the securities with a 'True'
    # value in the low_returns column.
    context.long_secs = context.output[context.output['low_returns']]

    # Sets the list of securities we want to short as the securities with a 'True'
    # value in the high_returns column.
    context.short_secs = context.output[context.output['high_returns']]

    # A list of the securities that we want to order today.
    context.security_list =
context.long_secs.index.union(context.short_secs.index).tolist()

    # A set of the same securities, sets have faster lookup.
    context.security_set = set(context.security_list)

def assign_weights(context):
    """
```

```
    Assign weights to long and short target positions.
    """

    # Set the allocations to even weights for each long position, and even weights
    # for each short position.
    context.long_weight = context.long_leverage / len(context.long_secs)
    context.short_weight = context.short_leverage / len(context.short_secs)

def rebalance(context,data):
    """
    This rebalancing function is called according to our schedule_function
settings.
    """

    assign_weights(context)

    # For each security in our universe, order long or short positions according
    # to our context.long_secs and context.short_secs lists.
    for stock in context.security_list:
        if data.can_trade(stock):
            if stock in context.long_secs.index:
                order_target_percent(stock, context.long_weight)
            elif stock in context.short_secs.index:
                order_target_percent(stock, context.short_weight)

    # Sell all previously held positions not in our new context.security_list.
    for stock in context.portfolio.positions:
        if stock not in context.security_set and data.can_trade(stock):
            order_target_percent(stock, 0)

    # Log the long and short orders each week.
    log.info("This week's longs: "+", ".join([long_.symbol for long_ in
context.long_secs.index]))
    log.info("This week's shorts: "  +", ".join([short_.symbol for short_ in
context.short_secs.index]))


def record_vars(context, data):
    """
    This function is called at the end of each day and plots certain variables.
    """

    # Check how many long and short positions we have.
    longs = shorts = 0
    for position in context.portfolio.positions.itervalues():
        if position.amount > 0:
            longs += 1
        if position.amount < 0:
            shorts += 1

    # Record and plot the leverage of our portfolio over time as well as the
    # number of long and short positions. Even in minute mode, only the end-of-day
    # leverage is plotted.
    record(leverage = context.account.leverage, long_count=longs,
short_count=shorts)


def handle_data(context,data):
```

```
    pass
```

27. Data Set

From https://www.quantopian.com/data

- Stock, Option, Future, Index, Economic
    - Quandl https://www.quandl.com/browse?idx=database-browser

- Analyst Estimates, Consensus Estimates
    - Estimize https://www.estimize.com/api

- Fundamental
    - EPS, Revenue, and Future Earnings Dates -- Benzinga http://cloud.benzinga.com/corporate-and-market-financial-data/

- Company Events
    - Spin-Offs, Lawsuit, Earnings Releases, Buyback Authorizations -- EventVestor http://www.eventvestor.com/

- Sentiment:
    - StockTweets Trader Mood -- PsychSignal https://psychsignal.com/hivemind/,
    - Sentiment Analysis -- SentDex http://sentdex.com/financial-analysis
    - News Sentiment -- Accern https://www.accern.com/

28. Options Trading Strategies

- 4 types: long call (outlook bullish, like movement), short call, long put (bearish), short put
- Greek:
    - moneyness: delta call—(0,1), put—(0,-1); OTM—(0,0.5), ITM—(0.5,1) probability of ending in the money
    - + gamma => buyer want underlying price to move, gamma increases as time run out
    - time decay: OTM option extrinsic value decays faster than ITM, bad for buyers; theta increases as time running out
    - implied volatility decreases as K increases; vega higher ATM, long expiry; buyers outlook higher vix (+ vega)
- Option Adjustment
    - Open interest => liquidity, find support resistance
    - delta neutral, outlook changed
        - Adjust long call => long strangle, bull call spread
        - Adjust short call => short strangle, bull call, bear call spread
        - Adjust long put => long strangle, bear put spread
        - Adjust short put => short strangle, bull put spread
- Other strategies
    - Butterfly = long ls put + short ms put + short ms put + long hs put
    - Iron Condor = long ls put + short ms put + short ms call + long hs call

Reference

29. Books

- John. C. Hull, *Options, Futures, and Other Derivatives* (with Excel code), 6th edition, Prentice Hall, 2006
- Paul Glasserman, *Monte Carlo Methods in Financial Engineering*, Springer 2013, Columbia Business School
- Benjamin Graham, *The Intelligent Investor*, Columbia Business School
- Mark. S. Joshi, *C++ Design Patterns and Derivatives Pricing* (with C++ code), 2nd edition, Wiley, 2008
- Paolo Brandimarte, *Handbook in Monte Carlo Simulation: Applications in Financial Engineering, Risk Management, and Economics* (with R & Matlab code), Wiley, 2014.
- Robert Shumway, David Stoffer, *Time Series Analysis and Its Applications* (with R code), Springer 2010
- CFA I, II curriculum: Quantitative Method, Equity, Fixed Income, Derivatives, and Portfolio Management.

30. Courses

- Martin Haugh, *Monte Carlo Simulation*, Columbia University
- Robert Kohn, Steve Allen, *Derivative Securities*, New York University
- Marco Avellaneda, *Risk and Portfolio Management with Econometrics*, New York University
- Peter Carr, Bruno Dupire, *Continuous Time Finance*, New York University
- Leonid Kogan, *Analytics of Finance* (with M code), MIT
- Peter Kempthorne, Choongbum Lee, Vasily Strela, Jake Xia, *Topics in Mathematics with Applications in Finance* (with R code), MIT
- Cliburn Chan, Janice McCarthy, *Statistical Programming* (with Python code), Duke University

31. On-line Course

- Jose Portilla, *Learning Python for Data Analysis and Visualization* (with Python code), Udemy
- Martin, *Time Series Analysis and Forecasting in R*, Udemy
- Martin, *Statistics in R*, Udemy
- Martin Haugh, Garud Iyengar, *Financial Engineering and Risk Management Part I*, Coursera, Columbia University
- Martin Haugh, Garud Iyengar, *Financial Engineering and Risk Management Part II*, Coursera, Columbia University
- Eric Zivot, *Introduction to Computational Finance and Financial Econometrics* (with R code), Coursera, University of Washington
- Brian Caffo, *Statistical Inference*, Coursera, Johns Hopkins University

- Brian Caffo, *Regression Models*, Coursera, Johns Hopkins University
- Jeff Leek, *Practical Machine Learning*, Coursera, Johns Hopkins University

32. Data

- http://www.cboe.com/
- http://www.idc.com/
- https://research.stlouisfed.org/
- http://ichart.finance.yahoo.com/table.csv?s=IBM

33. Competition & Conference

- WorldQuant Alphathon 2016
- QuantoPian Hackathon 2016
- Princeton Chicago Quant Trading Conference