```
# lab7.r                        script file for lab7 calculations
#
# author: Eric Zivot
# created: October 20, 2003
# revised: July 17, 2012
#
# comments:

options(digits=4, width=70)

# make sure packages are installed prior to loading them
library(PerformanceAnalytics)
library(zoo)
library(boot)
library(tseries)

# get monthly adjusted closing price data on VBLTX, FMAGX and SBUX from Yahoo
# using the tseries function get.hist.quote(). Set sample to Sept 2005 through
# Sep 2010. Note: if you are not careful with the start and end dates
# or if you set the retclass to "ts" then results might look weird

# get the last five years of monthly adjusted closing prices from Yahoo!
VBLTX.prices = get.hist.quote(instrument="vbltx", start="2005-09-01",
                             end="2010-09-30", quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")
# change class of time index to yearmon which is appropriate for monthly data
# index() and as.yearmon() are functions in the zoo package
#
index(VBLTX.prices) = as.yearmon(index(VBLTX.prices))

class(VBLTX.prices)
colnames(VBLTX.prices)
start(VBLTX.prices)
end(VBLTX.prices)

FMAGX.prices = get.hist.quote(instrument="fmagx", start="2005-09-01",
                             end="2010-09-30", quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")
index(FMAGX.prices) = as.yearmon(index(FMAGX.prices))

SBUX.prices = get.hist.quote(instrument="sbux", start="2005-09-01",
                             end="2010-09-30", quote="AdjClose",
                             provider="yahoo", origin="1970-01-01",
                             compression="m", retclass="zoo")
index(SBUX.prices) = as.yearmon(index(SBUX.prices))

# create merged price data
lab4Prices.z = merge(VBLTX.prices, FMAGX.prices, SBUX.prices)
# rename columns
colnames(lab4Prices.z) = c("VBLTX", "FMAGX", "SBUX")

# calculate cc returns as difference in log prices
lab4Returns.z = diff(log(lab4Prices.z))

#
# 3. Create timePlots of data
#
```

```
plot(lab4Returns.z, plot.type="single", lty=1:3, col=1:3, lwd=2)
legend(x="bottomleft", legend=colnames(lab4Returns.z), lty=1:3, col=1:3, lwd=2)
abline(h=0)
title("Monthly cc returns")


#
# 4. Create matrix of return data and compute pairwise scatterplots
#

ret.mat = coredata(lab4Returns.z)
colnames(ret.mat)
head(ret.mat)
VBLTX = ret.mat[,"VBLTX"]
FMAGX = ret.mat[,"FMAGX"]
SBUX = ret.mat[,"SBUX"]
pairs(ret.mat, col="blue")

#
# 5. Compute estimates of CER model parameters
#
muhat.vals = apply(ret.mat, 2, mean)
muhat.vals
sigma2hat.vals = apply(ret.mat, 2, var)
sigma2hat.vals
sigmahat.vals = apply(ret.mat, 2, sd)
sigmahat.vals
cov.mat = var(ret.mat)
cov.mat
cor.mat = cor(ret.mat)
cor.mat
covhat.vals = cov.mat[lower.tri(cov.mat)]
rhohat.vals = cor.mat[lower.tri(cor.mat)]
names(covhat.vals) <- names(rhohat.vals) <-
c("VBLTX,FMAGX","VBLTX,SBUX","FMAGX,SBUX")
covhat.vals
rhohat.vals

# summarize the CER model estimates
cbind(muhat.vals,sigma2hat.vals,sigmahat.vals)
cbind(covhat.vals,rhohat.vals)

# plot mean vs. sd values
plot(sigmahat.vals, muhat.vals, pch=1:3, cex=2, col=1:3,
     ylab = "mean", xlab="sd (risk)")
abline(h=0)
legend(x="topright", legend=names(muhat.vals), pch=1:3, col=1:3, cex=1.5)

#
# 6. Compute stndard errors for estimated parameters
#

# compute estimated standard error for mean
nobs = nrow(ret.mat)
nobs
se.muhat = sigmahat.vals/sqrt(nobs)
se.muhat
# show estimates with SE values underneath
rbind(muhat.vals,se.muhat)

# compute approx 95% confidence intervals
```

```
mu.lower = muhat.vals - 2*se.muhat
mu.upper = muhat.vals + 2*se.muhat
cbind(mu.lower,mu.upper)

# compute estimated standard errors for variance and sd
se.sigma2hat = sigma2hat.vals/sqrt(nobs/2)
se.sigma2hat
se.sigmahat = sigmahat.vals/sqrt(2*nobs)
se.sigmahat

rbind(sigma2hat.vals,se.sigma2hat)
rbind(sigmahat.vals,se.sigmahat)

# compute approx 95% confidence intervals
sigma2.lower = sigma2hat.vals - 2*se.sigma2hat
sigma2.upper = sigma2hat.vals + 2*se.sigma2hat
cbind(sigma2.lower,sigma2.upper)

sigma.lower = sigmahat.vals - 2*se.sigmahat
sigma.upper = sigmahat.vals + 2*se.sigmahat
cbind(sigma.lower,sigma.upper)

# compute estimated standard errors for correlation
se.rhohat = (1-rhohat.vals^2)/sqrt(nobs)
se.rhohat
rbind(rhohat.vals,se.rhohat)

# compute approx 95% confidence intervals
rho.lower = rhohat.vals - 2*se.rhohat
rho.upper = rhohat.vals + 2*se.rhohat
cbind(rho.lower,rho.upper)

#
# 7. Compute 5% and 1% Value at Risk
#

# function to compute Value-at-Risk
# note: default values are selected for
# the probability level (p) and the initial
# wealth (w). These values can be changed
# when calling the function. Highlight the entire
# function, right click and select run line or selection
Value.at.Risk = function(x,p=0.05,w=100000) {
        x = as.matrix(x)
        q = apply(x, 2, mean) + apply(x, 2, sd)*qnorm(p)
        VaR = (exp(q) - 1)*w
        VaR
}

# 5% and 1% VaR estimates based on W0 = 100000

Value.at.Risk(ret.mat,p=0.05,w=100000)
Value.at.Risk(ret.mat,p=0.01,w=100000)

################################################################################
# Hypothesis Testing
################################################################################

#
# 8. Test H0: mu = 0 vs. H1: mu /= 0
#
```

```
?t.test
t.test(lab4Returns.z[,"VBLTX"])
t.test(lab4Returns.z[,"FMAGX"])
t.test(lab4Returns.z[,"SBUX"])

#
# 9. Test H0: rho_ij = 0 vs. H1: rho_ij /= 0
#
?cor.test
# VBLTX,FMAGX
cor.test(x=lab4Returns.z[,"VBLTX"], y=lab4Returns.z[,"FMAGX"])
# VBLTX,SBUX
cor.test(x=lab4Returns.z[,"VBLTX"], y=lab4Returns.z[,"SBUX"])
# FMAGX,SBUX
cor.test(x=lab4Returns.z[,"FMAGX"], y=lab4Returns.z[,"SBUX"])

#
# 10. Test H0: returns are normal vs. H1: returns are not normal
#

library(tseries)
?jarque.bera.test

jarque.bera.test(lab4Returns.z[,"VBLTX"])
jarque.bera.test(lab4Returns.z[,"FMAGX"])
jarque.bera.test(lab4Returns.z[,"SBUX"])

#
# 11. 24 month rolling estimates of mu and sd
#

# rolling analysis for VBLTX
roll.mu.VBLTX = rollapply(lab4Returns.z[,"VBLTX"],
                          FUN=mean, width = 24, align="right")

roll.sd.VBLTX = rollapply(lab4Returns.z[,"VBLTX"],
                                  FUN=sd, width = 24, align="right")

plot(merge(roll.mu.VBLTX,roll.sd.VBLTX,lab4Returns.z[,"VBLTX"]), plot.type="single",
     main="24-month rolling means and sds for VBLTX", ylab="Percent per month",
     col=c("blue","red","black"), lwd=2)
abline(h=0)
legend(x="bottomleft", legend=c("Rolling means","Rolling sds", "VBLTX returns"),
       col=c("blue","red","black"), lwd=2)

#rolling analysis for FMAGX
roll.mu.FMAGX = rollapply(lab4Returns.z[,"FMAGX"],
                             FUN=mean, width = 24,
                             align="right")
roll.sd.FMAGX = rollapply(lab4Returns.z[,"FMAGX"],
                                FUN=sd, width = 24,
                              align="right")
plot(merge(roll.mu.FMAGX,roll.sd.FMAGX,lab4Returns.z[,"FMAGX"]), plot.type="single",
     main="24-month rolling means and sds for FMAGX",ylab="Percent per month",
     col=c("blue","red","black"), lwd=2)
abline(h=0)
legend(x="bottomleft", legend=c("Rolling means","Rolling sds", "FMAGX returns"),
       col=c("blue","red","black"), lwd=2)

# rolling analysis for SBUX
roll.mu.SBUX = rollapply(lab4Returns.z[,"SBUX"],
```

```
                             FUN=mean, width = 24,
                             align="right")
roll.sd.SBUX = rollapply(lab4Returns.z[,"SBUX"],
                                FUN=sd, width = 24,
                                align="right")
plot(merge(roll.mu.SBUX,roll.sd.SBUX,lab4Returns.z[,"SBUX"]), plot.type="single",
     main="24-month rolling means and sds for SBUX",ylab="Percent per month",
     col=c("blue","red","black"), lwd=2)
abline(h=0)
legend(x="bottomleft", legend=c("Rolling means","Rolling sds", "SBUX returns"),
       col=c("blue","red","black"), lwd=2)

# rolling correlation estimates
rhohat = function(x) {
        cor(x)[1,2]
}

# compute rolling estimates b/w VBLTX and FMAGX
roll.rhohat.VBLTX.FMAGX = rollapply(lab4Returns.z[,c("VBLTX","FMAGX")],
                       width=24,FUN=rhohat, by.column=FALSE,
                       align="right")
class(roll.rhohat.VBLTX.FMAGX)
roll.rhohat.VBLTX.FMAGX[1:5]

plot(roll.rhohat.VBLTX.FMAGX, main="Rolling Correlation b/w VBLTX and FMAGX",
     lwd=2, col="blue", ylab="rho.hat")
abline(h=0)

# compute rolling estimates b/w VBLTX and SBUX
roll.rhohat.VBLTX.SBUX = rollapply(lab4Returns.z[,c("VBLTX","SBUX")],
                       width=24,FUN=rhohat, by.column=FALSE,
                       align="right")
class(roll.rhohat.VBLTX.SBUX)
roll.rhohat.VBLTX.SBUX[1:5]

plot(roll.rhohat.VBLTX.SBUX, main="Rolling Correlation b/w VBLTX and SBUX",
     lwd=2, col="blue", ylab="rho.hat")
abline(h=0)

# compute rolling estimates b/w FMAGX and SBUX
roll.rhohat.FMAGX.SBUX = rollapply(lab4Returns.z[,c("FMAGX","SBUX")],
                       width=24,FUN=rhohat, by.column=FALSE,
                       align="right")
class(roll.rhohat.FMAGX.SBUX)
roll.rhohat.FMAGX.SBUX[1:5]

plot(roll.rhohat.FMAGX.SBUX, main="Rolling Correlation b/w FMAGX and SBUX",
     lwd=2, col="blue", ylab="rho.hat")
abline(h=0)

#
# 12. Evaluate bias and SE formulas using Monte Carlo
#

# generate 1000 samples from CER and compute sample statistics

mu = muhat.vals["FMAGX"]
sd = sigmahat.vals["FMAGX"]
n.obs = 60
set.seed(123)
n.sim = 1000
```

```
sim.means = rep(0,n.sim)
sim.vars = rep(0,n.sim)
sim.sds = rep(0,n.sim)
for (sim in 1:n.sim) {
        sim.ret = rnorm(n.obs,mean=mu,sd=sd)
        sim.means[sim] = mean(sim.ret)
        sim.vars[sim] = var(sim.ret)
        sim.sds[sim] = sqrt(sim.vars[sim])
}

par(mfrow=c(2,2))
hist(sim.means,xlab="mu hat", col="slateblue1")
abline(v=mu, col="white", lwd=2)
hist(sim.vars,xlab="sigma2 hat", col="slateblue1")
abline(v=sd^2, col="white", lwd=2)
hist(sim.sds,xlab="sigma hat", col="slateblue1")
abline(v=sd, col="white", lwd=2)
par(mfrow=c(1,1))

#
# 13. compute MC estimates of bias and SE
#

c(mu, mean(sim.means))
mean(sim.means) - mu
c(sd^2, mean(sim.vars))
mean(sim.vars) - sd^2
c(sd, mean(sim.sds))
mean(sim.sds) - sd

# compute MC SE value and compare to SE calculated from simulated data

c(se.muhat["FMAGX"], sd(sim.means))
c(se.sigma2hat["FMAGX"], sd(sim.vars))
c(se.sigmahat["FMAGX"], sd(sim.sds))



#
# 14. bootstrapping SE for mean, variance, sd and correlation
#

?boot
# note: boot requires user-supplied functions that take
# two arguments: data and an index. The index is created
# by the boot function and represents random resampling with
# replacement

# function for bootstrapping sample mean
mean.boot = function(x, idx) {
# arguments:
# x             data to be resampled
# idx           vector of scrambled indices created by boot() function
# value:
# ans           mean value computed using resampled data
     ans = mean(x[idx])
     ans
}

VBLTX.mean.boot = boot(VBLTX, statistic = mean.boot, R=999)
class(VBLTX.mean.boot)
```

```
names(VBLTX.mean.boot)

# print, plot and qqnorm methods
VBLTX.mean.boot
se.muhat["VBLTX"]

# plot bootstrap distribution and qq-plot against normal
plot(VBLTX.mean.boot)


# compute bootstrap confidence intervals from normal approximation
# basic bootstrap method and percentile intervals
boot.ci(VBLTX.mean.boot, conf = 0.95, type = c("norm","perc"))

#
# boostrap SD estimate
#
# function for bootstrapping sample standard deviation
sd.boot = function(x, idx) {
# arguments:
# x             data to be resampled
# idx           vector of scrambled indices created by boot() function
# value:
# ans           sd value computed using resampled data
     ans = sd(x[idx])
     ans
}

VBLTX.sd.boot = boot(VBLTX, statistic = sd.boot, R=999)
VBLTX.sd.boot
se.sigmahat["VBLTX"]

# plot bootstrap distribution
plot(VBLTX.sd.boot)

# compute confidence intervals
boot.ci(VBLTX.sd.boot, conf=0.95, type=c("norm", "basic", "perc"))

# bootstrap correlation

# function to compute correlation between 1st 2 variables in matrix
rho.boot = function(x.mat, idx) {
# x.mat n x 2 data matrix to be resampled
# idx           vector of scrambled indices created by boot() function
# value:
# ans           correlation value computed using resampled data

        ans = cor(x.mat[idx,])[1,2]
        ans
}
VBLTX.FMAGX.cor.boot = boot(ret.mat[,c("VBLTX","FMAGX")],
                            statistic=rho.boot, R = 999)
VBLTX.FMAGX.cor.boot
se.rhohat[1]

# plot bootstrap distribution
plot(VBLTX.FMAGX.cor.boot)

# bootstrap confidence intervals
boot.ci(VBLTX.FMAGX.cor.boot, conf=0.95, type=c("norm", "perc"))
```

```
#
# 15. Bootstrap VaR
#

# 5% Value-at-Risk
ValueAtRisk.boot = function(x, idx, p=0.05, w=100000) {
# x.mat data to be resampled
# idx           vector of scrambled indices created by boot() function
# p             probability value for VaR calculation
# w             value of initial investment
# value:
# ans           Value-at-Risk computed using resampled data

        q = mean(x[idx]) + sd(x[idx])*qnorm(p)
        VaR = (exp(q) - 1)*w
        VaR
}

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