LAB MANUAL:06

MAT 3208: TIME SERIES

DATE:

COVARIANCE AND CORRELATION

`mean(stockprice)`: mean
`sd(stockprice)`: standard deviation
`cov(stockprice_A, stockprice_B)`:Covariance

Correlation

- standardized version of covariance
- +1: perfectly positive linear relationship
- -1: perfectly negative linear relationship
- 0: no linear association
- `cor(stockprice_A, stockprice_B)`
- `cov(stockprice_A,stockprice_B)/ (sd(stockprice_A)*sd(stockprice_B))`

AUTOCORRELATION

- Autocorrelations or lagged correlations are used to assess whether a time series is dependent on its past.
- For a time series `x` of length `n` we consider the `n-1` pairs of observations one time unit apart. The first such pair is `(x[2],x[1])`, and the next is `(x[3],x[2])`. Each such pair is of the form `(x[t],x[t-1])` where t is the observation index, which we vary from 2 to n in this case. The lag-1 autocorrelation of `x` can be estimated as the sample correlation of these `(x[t], x[t-1])` pairs.
- In general, we can manually create these pairs of observations. First, create two vectors, `x_t0` and `x_t1`, each with length `n-1`, such that the rows correspond to `(x[t], x[t-1])` pairs. Then apply the `cor()` function to estimate the lag-1 autocorrelation.
- The **`acf()`** ,Applying `acf(..., lag.max = 1, plot = FALSE)` to a series `x` automatically calculates the lag-1 autocorrelation.
- The two estimates differ slightly as they use slightly different scaling in their calculation of sample covariance, 1/(n-1) versus 1/n. Although the latter would provide a biased estimate, **it is preferred in time series analysis**, and the resulting autocorrelation estimates only differ by a factor of (n-1)/n.

Instructions

```
# Define x_t0 as x[-1]
x_t0 <-
# Define x_t1 as x[-n]
x_t1 <-
# Confirm that x_t0 and x_t1 are (x[t], x[t-1]) pairs
head(cbind(x_t0, x_t1))
# Plot x_t0 and x_t1
plot(___, ___)
# View the correlation between x_t0 and x_t1
cor(___, ___)
# Use acf with x
acf(___, lag.max = ___, plot = ___)
# Confirm that difference factor is (n-1)/n
cor(x_t1, x_t0) * (n-1)/n
```

Autocorrelations can be estimated at many lags to better assess how a time series relates to its past. We are typically most interested in how a series relates to its most recent past.

AUTOREGRESSIONS

Suppose we consider the white noise series ω_t of previous example as input and calculate the output using the second-order equation.

$$x_t = x_{t-1} - .9x_{t-2} + \omega_t$$

successively for t = 1, 2, . . . , 500.

Above equation represents a regression or prediction of the current value x_t of a time series as a function of the past two values of the series, and, hence, the term autoregression is suggested for this model. A problem with startup values exists here because above equation also depends on the initial conditions x_0 and x_{-1}.

```
w = rnorm(550,0,1) # 50 extra to avoid startup problems
x = filter(w, filter=c(1,-.9), method="recursive")[-(1:50)]
plot.ts(x, main="autoregression")
```

PROBLEM 01

use arima.sim to simulate 100 observations of an AR model with slopes. (x→ 0.5, y→0.9, z → -0.75). Then plot that simulated data and calculate the ACF for those three ts objects.

```
# Simulate an AR model with 0.5 slope
x <- arima.sim(model = list(ar = 0.5), n = 100)
# Simulate an AR model with 0.9 slope
y <-
# Simulate an AR model with -0.75 slope
```

```
z <-
# Plot your simulated data
plot.ts(cbind(x, y, z))
# Calculate the ACF for x
acf(x)
# Calculate the ACF for y

# Calculate the ACF for z
```

Persistence and anti-persistence

- Persistence is defined by a high correlation between an observation and its lag.
- anti-persistence is defined by a large amount of variation between an observation and its lag.

RANDOM WALK(RW) AND AUTOREGRESSIVE(AR) MODEL

The random walk (RW) model is a special case of the autoregressive (AR) model, in which the slope parameter is equal to 1. RW model is not stationary and exhibits very strong persistence. Its sample autocovariance function (ACF) also decays to zero very slowly, meaning past values have a long lasting impact on current values.
The slope in an AR model **can range from -1 to 1**. As the slope gets closer to 1, the model shows higher persistence, i.e., it shows higher correlation with previous values. Also, the higher the slope, the slower is the decay of ACF to 0.

PROBLEM 02

Compare Random Walk(RW) and Autoregressive(AR) model.

PROBLEM 03

Random walk with drift model is given by the following equation.

$$x_t = \delta + x_{t-1} + w_t$$

for t = 1, 2, . . ., with initial condition $x_0 = 0$, and where w_t is white noise (The constant δ is called the drift, and when δ = 0, is called simply a **random walk**).
Use the following codes to generate plots for the models with δ = 0 and .2(with σ_w = 1) and a straight line. Comment on the graphs you plotted.

```
set.seed(154) # so you can reproduce the results
w = rnorm(200,0,1); x = cumsum(w) # two commands in one line
wd = w +.2; xd = cumsum(wd)
plot.ts(xd, ylim=c(-5,55), main="random walk")
lines(x); lines(.2*(1:200), lty="dashed")
```

PROBLEM 04

Use `arima.sim()` to simulate 200 observations from an AR model with slope `0.9`. Save this to `x`. Use `ts.plot()` to plot `x` and use `acf()` to view its sample ACF. Now do the same from an AR model with slope `0.98`. Save this to`y`. Now do the same

from a RW model (`z`), and compare the time series and sample ACFs generated by these three models.

```
# Simulate and plot AR model with slope 0.9
x <- arima.sim(model = ___, n = ___)
ts.plot(___)
acf(___)
# Simulate and plot AR model with slope 0.98
y <-
ts.plot(___)
acf(___)
# Simulate and plot RW model
z <-
ts.plot(___)
acf(___)
```

PROBLEM 05

Generate n = 100 observations from the autoregression $x_t = -.9x_{t-2} + w_t$ with $\sigma_w = 1$, using the method described in previous example. Next, apply the moving average filter $v_t = (x_t + x_{t-1} + x_{t-2} + x_{t-3})/4$ to x_t, the data you generated. Now plot x_t as a line and superimpose v_t as a dashed line. Comment on the behavior of x_t and how applying the moving average filter changes that behavior. [Hints: Use v = filter(x, rep(1/4, 4), sides = 1) for the filter and note that the R code in problem 03 may be of help on how to add lines to existing plots.]