R Applications for non-stationary and persistent time series models

Introductory Remarks

This file builds on the previous R Activities. If you have not yet gone through those files, please make sure to do so before tackling this R Activity.

Trends and Seasonality in Time Series Data

In the previous R activity, we learned how to use the "dynlm" function to run a FDL or AR(p) model. From the lectures we know that time series are often display a trend or seasonal fluctuations. We now turn to the question how to account for such data using R.

Accounting for Trends

Time-series data often contain trends. To see this, load and time-set the HSEINV dataset on housing investments and housing prices from 1947 to 1988.

```
data("hseinv") #load data
hseinv_ts <- zoo(hseinv, order.by = hseinv$year) # ts data
```

Suppose we are interested in regressing log housing investments on log housing prices. A naive approach might look like this

$$log(invpc_t) = \alpha_0 + \alpha_1 log(price_t) + \epsilon_t$$

Implemented in R, this looks as follows

```
house_reg <- dynlm(log(invpc) ~ log(price), data=hseinv_ts)
summary(house_reg)
```

Notice that the effect of log prices on log investments is positive and significant at the 5% level. It implies that a 1% increase in prices is associated with a 1.241% increase in investments. Can we trust these numbers? No! If, for example, there are time trends in our variables, then we are worried that we might be accidentally attributing this time trend as a causal effect of log prices on log investments. To analyze this, we can simply look at the variables, i.e. type

```
plot(log(hseinv_ts$price)) # plot log price
plot(log(hseinv_ts$invpc)) # plot log investment
```

Indeed, we see a trend. To take this into account, we can add a time trend to our regression equation, i.e.

$$log(invpc_t) = \alpha_0 + \alpha_1 log(price_t) + \gamma t + \epsilon_t$$

In R, we do this as follows

```
house_reg2 <- dynlm(log(invpc) ~ log(price) + trend(hseinv_ts), data=hseinv_ts)
summary(house_reg2)
```

In order to include this time trend, we can type "trend(hseinv_ts)." Given that we time-set the dataset, R will know to include the right time trend into this regression. Notice that the coefficient on log price is no longer significant, thus showing how dangerous it can be to forget to include time trends into our regressions.

Accounting for Seasonality

Consider the BARIUM dataset on Chinese imports of barium chloride from the previous R activity, time-set the data, and plot the Chinese imports, i.e. type

```
data("barium") # loading the barium data
barium_ts <- ts(barium, start = c(1978, 2), frequency = 12) # ts full data
imports<- ts(barium$chnimp, start=c(1978,2), frequency=12) # ts only one variable
plot(imports) # plot imports
```

Note that we can choose to timeset either the full dataset or just one variable. We will need the full dataset in a bit, but to just plot the imports over time, time-setting the "chnimp" variable alone would suffice. As you can see, this plot exhibits clear seasonal swings. As you learned in the lecture, we want to take this seasonality into account when running our regressions.

To take this seasonality into account in this regression, consider running a regression of Chinese barium chloride imports on a chemical production index, gasoline production, an exchange rate index, a set of dummies, as well as a set of monthly dummies to take into account the seasonality.

```
barium_reg <- dynlm(log(chnimp) ~ log(chempi) + log(gas) + log(rtwex) + befile6
                    + affile6 + afdec6 + season(barium_ts), data=barium_ts)
summary(barium_reg)
```

Once again, R makes taking seasonality into account very simple. As you can see, we just add "season(barium_ts)" to the regression and R will automatically include the month dummies to account for the seasonality. As you can see in the output, none of these dummies are significant (except the April dummy, which is significant at the 10% level). To test whether they are jointly significant, we could conduct an F-test, i.e.

```
linearHypothesis(barium_reg, c("season(barium_ts)Feb=0", "season(barium_ts)Mar=0",
                               "season(barium_ts)Apr=0", "season(barium_ts)May=0",
                               "season(barium_ts)Jun=0", "season(barium_ts)Jul=0",
                               "season(barium_ts)Aug=0", "season(barium_ts)Sep=0",
                               "season(barium_ts)Oct=0", "season(barium_ts)Nov=0",
                               "season(barium_ts)Dec=0"))
```

As you can see, the F-statistic is 0.8559 with a p-value of 0.5852. We therefore fail to reject the null hypothesis that the monthly dummies are jointly zero.

Random Walks and First Differences

As you've learned in the lectures, time series data are often dependent and persistent. A great example of such a highly persistent and dependent process is a random walk, i.e.

$$y_t = y_{t-1} + \epsilon_t$$

where ϵ_t are iid shocks with mean zero. Note that this is basically an AR(1) process, where the intercept is zero and the parameter on y_{t-1} is equal to 1. From the lectures and the provided R scripts, you have been told how to rewrite this random walk in terms of y_0 and the sum of $\epsilon_1, \epsilon_2, ..., \epsilon_t$, as well as how to simulate such a process in R.

We know that this persistence and dependence over time can cause issues in our regressions. One way to deal with this persistence and dependence is to difference the time series. Consider the following random walk, this time with an added drift

$$y_t = \beta_0 + y_{t-1} + \epsilon_t$$

Subtract y_{t-1} from both sides to get

$$\Delta y_t = y_t - y_{t-1} = \beta_0 + \epsilon_t$$

This, it turns out, is now an iid process with mean β_0. To see what this looks like in R, let's simulate the differenced random walk, i.e.

```
set.seed(348546) # set seed
plot(c(0,50), c(2,2), type="l", lwd=2, ylim=c(-1, 5)) # initialize graph
for (r in 1:30){ # start loop to simulate 30 random walks
  eps <- rnorm(50) # simulate epsilons from std normal distr (get 50 epsilons)
  y <- ts(cumsum(2 + eps)) # y, the time series, is just the sum of all the epsilons
  Dy <- diff(y) # take difference
  lines(Dy, col=gray(.6)) # add this random walk into the plot above as a gray line
}
```

As you can see, the code is similar to the one when you simulate a random walk, with the added difference that after simulating the random walk with drift, we difference it with the "diff" command. Note that the "diff" command only works because we have time-set the data in the row above. As you can see in the resulting graph, we now have a process that looks stationary.

Let's now try to apply this idea of differencing the data to the FERTIL3 data.

```
data("fertil3")
fertil3_ts <- zoo(fertil3, order.by = fertil3$year) # ts data
```

Recall that this is the dataset where we regress the fertility rate on personal tax exemptions, its lags, as well as some controls. For simplicity, ignore the controls, and consider two regressions. First, let's regress the differenced fertility rate on the differences tax exemption variable. Second, let's add to the first regression two differenced lags of the tax exemption variable. In R, this looks as follows

```
fertil_reg1 <- dynlm(d(gfr) ~ d(pe), data=fertil3_ts)
fertil_reg2 <- dynlm(d(gfr) ~ d(pe) + L(d(pe)) + L(d(pe), 2), data=fertil3_ts)
summary(fertil_reg1)
summary(fertil_reg2)
```

Important for the purposes of R here is to see that in order to difference equations, we just include "d" into the "dynlm" function.

Testing for the Presence of a Unit Root

Given the issues associated with random walks (see above), we would like to have a way to test whether these issues are present in our data. A more formal way of putting this is that we'd like to test for the presence of a unit root. As you've seen in class, one way to do this is to use the (augmented) Dickey-Fuller (ADF) test.

Conceptually, if you want to test whether a time series y has a unit root, the ADF tells you to regress Δy_t on y_{t-1}, Δy_{t-1}, Δy_{t-2}, etc. To test the null hypothesis of a unit root, you then test whether the coefficient on y_{t-1} is equal to zero or not. Note that if you fail to reject the null hypothesis, this means that you have non-stationary data (i.e. you have a unit root).

In R, the "tseries" package contains a command called "adf.test," which implements this test. To see how we can implement it, recall that when we simulated the random walks above, we called the random walk "y" and the differenced random walk "Dy." From the lectures, we know that the former has a unit root and the latter should not have one. Let's see if an ADF test tells us the same.

```
adf.test(y, k=1) # ADF test
adf.test(Dy, k=1) # ADF test
```

Note that the option "k" is there for you to decide how many of the lagged differences (Δy_{t-1}, Δy_{t-2}, etc.) you want to include. The p-value of the ADF test on "y" is 0.2881, while the p-value of the ADF test on "Dy" is 0.01. This means that we fail to reject the null hypothesis of a unit root for "y" (i.e. we have a unit root) and we reject the null hypothesis of a unit root for "Dy" (i.e. we don't have a unit root). Our results, it seems, are as expected.