Lab-7

March 9, 2023

1 Spectral Analysis and Multivariate Analysis

This workshop is split into two parts:

- Spectral Analysis and
- Multivariate Analysis

There is an assignment at the end to complete.

1.1 Spectral analysis

In this part of the workshop we will look at methods for analysing time series data via techniques based on spectral analysis. We first simulate a collection of datasets of differing complexity. This code was taken from the course resource file: harmonic_regression.R

[ ]:
```
options(repr.plot.width=15, repr.plot.height=15) # makes plots bigger in the␣
 ↪webpage
```

[ ]:
```
library(TSA)
library(miscFuncs) # a great little package :-)

set.seed(1) # for reproducibility

n = 1000
P = 7
nP = 12
t = seq(0,P*nP,length.out=n) # time points over a "12 week" period

# d contains s1, c1, s2, c2, ... where s'k' = sin(2*pi*k*t/P) and c'k' =␣
 ↪cos(2*pi*k*t/P)
d = genharmonic(data.frame(t=t),tname="t",base=P,num=5) # generate harmonic␣
 ↪basis

print(head(d))
```

Let's look at some simulated harmonic series. Do this by sampling random coefficients a_k and b_k in the (truncated) Fourier series:

$$f(t) = \sum_{k=1}^{K} a_k \sin(2\pi k t/P) + b_k \cos(2\pi k t/P)$$

We'll sample the coefficients using a Gaussian, but they can be anything you choose, really (feel free to edit and see what the consequences are …)

[ ]:
```
nsamp = 5
for(i in 1:5){ # i will be the number of harmonic terms we'll include
    for(j in 1:nsamp){ # look at nsamp different simualtions
        sco = rnorm(i)
        cco = rnorm(i)
        if(i == 5 & j ==3){
            strue = sco
            ctrue = cco
        }
        oname = paste("s",i,j,sep="")
        d[[oname]] = 0 # initialise
        for (k in 1:i){
            d[[oname]] = d[[oname]] + sco[k]*d[[paste("s",k,sep="")]]
        }
        for (k in 1:i){
            d[[oname]] = d[[oname]] + cco[k]*d[[paste("c",k,sep="")]]
        }
        #plot(d[["t"]],d[[oname]],type="l",lwd=3,xlab="t",main = paste("Number␣
 ↪of Harmonic terms =",i))
        #readline("Press a key to contuinue")
    }

}

head(d)
```

Now look at one of the series and simulate a time series based on this

[ ]:
```
raw = ts(d$s14,frequency=n/(P*nP))
per = ts(raw + rnorm(n,0,0.6),frequency=n/(P*nP)) # one harmonic term, fourth␣
 ↪simuated series with some noise added
```

In the above, the `frequency` argument has been set to `n / (P*nP)` Recall that `P*nP = 7 * 12` i.e. the period length (set to 7 "days" here) multiplied by the number of "weeks", 12. This interval was divided into 1000 equal width intervals of width (7*12)/999 (because the tiime origin was zero in this case).

[ ]:
```
par(mfrow=c(1,2))
plot(d$t,raw,type="l",main="Raw Signal")
plot(d$t,per,type="l",main="Signal + Noise")
lines(lowess(d$t,per,f=0.05),col="red",lwd=3)
```

The **spectrum** of a series $\{X_t\}$ is defined as:

$$f(\omega) = \frac{1}{\pi} \sum_{k=-\infty}^{\infty} \gamma(k) e^{-ik\omega}.$$

I.e. the discrete Fourier transform of the autocovariance function. We can use the function `spectrum` in R to calculate this for a given series.

```
[ ]: par(mfrow=c(1,2))
     sraw = spectrum(raw)
     sper = spectrum(per)
```

The spectrum (or periodogram as it is also known) will have a marked peak in it at a particular frequency ω if the signal has periodic behaviour at that frequency too. Let's try to use the spectrum here to work out what ω is for our data.

```
[ ]: omega_raw = sraw$freq[which(sraw$spec==max(sraw$spec))]
     omega_raw
```

```
[ ]: omega_per = sper$freq[which(sper$spec==max(sper$spec))]
     omega_per
```

And we need to take the reciprocal of that to get back to the period:

```
[ ]: 1/omega_per
```

... so all looks good. Note that this method of looking up the maximum only works in this case because the peak is so strong that it stands out from the rest of the periodogram. For example, try running the following a few times:

```
[ ]: tst = ts(raw + rnorm(n,0,5),frequency=n/(P*nP))
     spectrum(tst)
```

Let's look at another series:

```
[ ]: raw1 = ts(d$s34,frequency=n/(P*nP))
     per1 = ts(raw1 + rnorm(n,0,0.6),frequency=n/(P*nP)) # three harmonic terms,␣
      ↪fourth simuated series with some noise added
     par(mfrow=c(1,2))
     plot(d$t,raw1,type="l",main="Raw Signal")
     plot(d$t,per1,type="l",main="Signal + Noise")
     lines(lowess(d$t,per1,f=0.05),col="red",lwd=3)
```

```
[ ]: par(mfrow=c(1,2))
     sraw = spectrum(raw1)
     sper = spectrum(per1)
```

And this time, there are three peaks early on in the periodogram in both cases. Let's try one more series, this time with five component terms ...

```
[ ]: raw2 = ts(d$s53,frequency=n/(P*nP))
     per2 = ts(raw2 + rnorm(n,0,0.6),frequency=n/(P*nP)) # five harmonic terms,␣
      ↪third simuated series with some noise added
     par(mfrow=c(1,2))
     plot(d$t,raw2,type="l",main="Raw Signal")
     plot(d$t,per2,type="l",main="Signal + Noise")
     lines(lowess(d$t,per2,f=0.05),col="red",lwd=3)
```

With spectra:

```
[ ]: par(mfrow=c(1,2))
     sraw = spectrum(raw2)
     sper = spectrum(per2)
```

Zooming in now:

```
[ ]: plot(sper$freq[1:100],log(sper$spec[1:100]),type="l",main="Zoom in on Spectrum")
     abline(v=1/c(7,7/2,7/3,7/4,7/5),lty="dashed",col="blue")
     # to confirm the periods detected in the spectrum of the raw series
```

Let's now simulate an ARMA(p,q) process for p = 5 and q = 3. If you need a reminder on simulating processes, then please revisit the earlier lab.

```
[ ]: mysigma = 1
     myrands = function(n){
         return(rnorm(n,0,mysigma)) # note this is dirty programming, but I want to␣
      ↪be able to access "mysigma" later
     }
     a = arima.sim(model=list(ar=c(0.5,0.2,0.1,0.05,0.01),ma=c(2,-0.7,-0.
      ↪2)),n=10000,rand.gen=myrands)
     plot(a)
```

In the lecture we cover that the spectrum of this process has form:

$$f(\omega) = \frac{\sigma^2}{2\pi}\frac{|\theta(\mathrm{e}^{-i\omega})|^2}{|\phi(\mathrm{e}^{-i\omega})|^2}$$

Now our ARMA(5,3) process has

$$\phi(z) = 1 - 0.5z - 0.2z^2 - 0.1z^3 - 0.05z^4 - 0.01z^5$$

and

$$\theta(z) = 1 + 2z - 0.7z^2 - 0.2z^3$$

Let's create some functions to evaluate these explicitly.

```
[ ]: cpow = function(z,p){ # a function for evaluating powers of complex numbers:
                           # only works with positive integer powers
         ans = z
         if(p==1){
             return(z)
         }
         else{
             for(i in 2:p){
                 ans = ans * z
             }
             return(ans)
         }
     }

     # Now some functions to evaluate $\phi(z)$ and $\theta(z)$ based on the above␣
      ↪coefficients
     phi = function(z){
         return(1 - 0.5*cpow(z,1) - 0.2*cpow(z,2) - 0.1*cpow(z,3) - 0.05*cpow(z,4) -␣
      ↪0.01*cpow(z,5))
     }

     theta = function(z){
         return(1 + 2*cpow(z,1) - 0.7*cpow(z,2) - 0.2*cpow(z,3))
     }
```

```
[ ]: eix = function(x){
         return(complex(real=cos(x),imaginary=sin(x)))
     }

     f = function(omega,sigma){
         return((sigma^2/(2*pi))*(Mod(theta(eix(-omega)))/Mod(phi(eix(-omega))))^2)
     }

     omegaseq = seq(0,pi,length.out=1000)
     plot(omegaseq,sapply(omegaseq,f,sigma=mysigma),type="l")
```

```
[ ]: sa = spectrum(a,fast=FALSE)
     lines(sa$freq,sapply(sa$freq,f,sigma=mysigma),col="red",lwd=3)
```

1.1.1 Exercise

Modify the above functions in order to plot an arbitrary spectral density of an ARMA(p,q) process i.e. so that you pass in the theoretical coefficients of the AR and MA models and have it make a plot of the process. Include an argument "add" to allow this theoretical line to be drawn on top of an empirical periodogram.

1.2 Multivariate analysis

In this part of the workshop we will look at statistical methods for analysing multiple time series data streams simultaneously. Multivariate analysis is a powerful tool for forecasting because it allows us to borrow strength across multiple co-evolving, co-related series.

1.2.1 Simulating and Anlaysing Vector Autoregressive Data

The VAR(p) model over $\mathbb{R}^d$ has the form:

$$Y_t = a + A_1 Y_{t-1} + A_2 Y_{t-2} + \cdots + A_p Y_{t-p} + e_t$$

where: - $Y_t = (y_{1t}, y_{2t}, \ldots, y_{dt})^T \in \mathbb{R}^d$ is an d-vector of d time series variables - $a \in \mathbb{R}^d$ is an d-vector of intercepts - For each i, A_i is a $d \times d$ matrix of coefficients - e_t is an d-vector of white noise

Let's simulate from a VAR(2) model with correlated noise. We'll then try to retrieve the parameters we used in the simulation by analysing the resulting dataset as if we did not know how it was constructed in the first place.

We'll first write a piece of code to sample from a generic multivariate normal density MVN(μ, Σ). Although there are software packages already implenting this, it is informative to understand how this is done in practice, from first principles (or close to them). As Brian Ripley once said: "you never really understand a computational method until you have successfully implemented it yourself".

We use the fact that:

$$e = \mu + \Sigma^{1/2} U \sim \text{MVN}(\mu, \Sigma),$$

where U are i.i.d. standard Gaussian and $\Sigma^{1/2}$ is the Cholesky decomposition of the matrix Σ.

In the example, we'll simulate from a tri-variate VAR(2) model.

```
[ ]: Sigma = matrix(c(1,0.5,0.25,0.5,2,-0.1,0.25,-0.1,3),3,3)
     Sigma
```

And compute the Cholesky. Note that in `R`, this needs to be transposed:

```
[ ]: cholSigma = t(chol(Sigma))
     cholSigma
```

```
[ ]: mvnsim = function(n,mu,sigma){  # this function will expect vector mu and␣
      ↪matrix sigma arguments.
                                     # n will be the number of realisations to␣
      ↪generate
         d = length(mu)
         ds = dim(sigma)
         if(ds[1]!=d |ds[2]!=d){ # a simple check
             stop(paste("Matrix sigma must be ",d," x ",d,sep=""))
         }
         U = matrix(rnorm(n*d),d,n) # the standard normals
```

```
    ch = t(chol(sigma)) # the Cholesky
    return(t(mu + ch%*%U)) # there are some tricks here to make things faster
}
```

[ ]:
```
# test it:
nrms = mvnsim(10000,c(0,0,0),Sigma)
pairs(nrms,pch=".")
```

[ ]:
```
cov(nrms) # retrive the empirical variance/covariance matrix
```

[ ]:
```
# More tests:
apply(mvnsim(10000,c(2,5,10),Sigma),2,mean)
```

We'll use this function to simulate the (white) noise process. We now need some matrices for the other parts of the model.

[ ]:
```
A1 = matrix(c( 0.5, -0.2, 0.05,
              -0.2,  0.5, -0.1,
              0.05, -0.1,  0.5),3,3)
A1
```

[ ]:
```
A2 = matrix(c(  0.25,   0.1, -0.025,
                 0.1,  0.25,  -0.05,
              -0.025, -0.05,   0.25),3,3)
A2
```

[ ]:
```
n = 10000
Y = matrix(0,n,3) # initialise Y, we'll set Y[1,] = Y[2,] = (0,0,0)
e = mvnsim(n,c(0,0,0),Sigma) # zero-mean indepent correlated-MVN variates
for(i in 3:n){
    Y[i,] = A1%*%Y[i-1,] + A2%*%Y[i-2,] + e[i,]
}
par(mfrow=c(3,1))
plot(Y[,1],type="l",main="Y, First Component")
plot(Y[,2],type="l",main="Y, Second Component")
plot(Y[,3],type="l",main="Y, Third Component")
```

Note that using this kind of method, you might need to use a burn-in period before the simulations reach stationarity. To see this, try editing the above simulation code to include an intercept term, a, as in the detail of the VAR$(\cdot)$ model above. For the purposes of the analysis below, we'll ignore this as n is much bigger than 2.

Now we have a dataset, we can use the `vars` package to analyse it. First some more exploratory plots ...

[ ]:
```
library(vars)
pairs(Y,pch=".")
```

```
[ ]: matplot(Y[1:200,],type="l") # zooming in
```

```
[ ]: Y1 = ts(Y[,1],start=0,frequency=1) # converting each series to a ts object
     Y2 = ts(Y[,2],start=0,frequency=1)
     Y3 = ts(Y[,3],start=0,frequency=1)
```

```
[ ]: y = cbind(Y1,Y2,Y3) # switching to lower case to avoid object duplicity
     str(y)
```

We next compute the tri-variate ACF:

```
[ ]: par(mfrow=c(3,3))

     for(i in 1:3){
         y1 = get(paste("Y",i,sep=""))
         for(j in 1:3){
             y2 = get(paste("Y",i,sep=""))
             if(i==j){
                 acf(y1,main=paste("ACF",paste("Y",i,sep="")))
             }
             else{
                 ccf(y1,y2,main=paste("CCF",paste("Y",i," and Y",j,sep="")))
             }
         }
     }
```

```
[ ]: par(mfrow=c(1,2))
     spectrum(y)
     spec.pgram(y,kernel("modified.daniell", c(100,100))) # to smooth out the noise
```

Now use the function `VARselect` from the package `vars` to select a best VAR model ...

```
[ ]: out = VARselect(y)
     out
```

```
[ ]: par(mfrow=c(2,2))
     plot(out$criteria[1,],main="AIC",type="l")
     plot(out$criteria[2,],main="HQ",type="l")
     plot(out$criteria[3,],main="SC",type="l")
     plot(out$criteria[4,],main="FPE",type="l")
```

- SC = Schwartz criterion, named after Gideon Schwartz, is another name for the BIC
- HQ = Hannan-Quinn criterion
- FPE = final prediction error.

And from this, we conclude that VAR(2) looks to be the best, though there is not much difference between this model and the VAR(3), according to AIC. Recall that the SC (equiv BIC) induces a more harsh penalty for additional parameters compared to the AIC, and it is clear from this diagnostic, that a model of order 2 is the best.

Let's fit that VAR(2) model:

[ ]:
```
mod = VAR(y,p=2)
summary(mod)
```

Lots of parameters! Note how the estimated noise matches the true value. We next produce forecasts, here using the default 10 steps ahead.

[ ]:
```
preds = predict(mod)
preds # careful here: someone has written a print method for this class ...
```

[ ]:
```
# and plot the forecasts:

par(mfrow=c(3,1))
for(i in 1:3){
    nm = paste("Y",i,sep="")
    plot(9951:10010,c(get(nm)[9951:10000],preds$fcst[[nm]][,1]),
                             ␣
 ↪col=c(rep("black",50),rep("red",10)),pch=19,ylim=c(-6,6),main=nm)
    lines(9951:10000,get(nm)[9951:10000])
    polygon(c(10001:10010,10010:
 ↪10001),c(preds$fcst[[nm]][,2],rev(preds$fcst[[nm]][,3])),col=rgb(0,0,1,alpha=0.
 ↪2),border=NA)
}
```

1.2.2 Exercise

Try using the first 8000 observations to train your VAR model. Then pretend you did not observe the last 2000 observations for the third component of the time series. Use the last 2000 observations of the first two components of the series to predict the third in this incompletely-observed interval and compare your predictions to the true value of the third component.

1.2.3 Exercise: EU Stock Price Data

Consider the EU Stockprice data, `EuStockMarkets`:

[ ]:
```
dim(EuStockMarkets)
```

[ ]:
```
head(EuStockMarkets)
```

Repeat the techniques used in the simulated data example to analyse this dataset and produce forecasts from an appropriate VAR(p) model.

1.3 Assignment

Upload a 2x2 set of plots showing forecasts with confidence bands for each of the EU stock considered: DAX, SMI, CAC, FTSE.