Time Series

Joshua Tegegne

05/12/2021

```
knitr::opts_chunk$set(warning=FALSE, message=FALSE) #download and install packages
if (!require(pacman)){install.packages("pacman", repos = "http://cran.us.r-project.org")}
```

```
## Warning: package 'pacman' was built under R version 4.0.5
```

```
pacman::p_load(stats, comprehenr, swdft, dplyr, ggplot2, scales,SynchWave, readxl, knitr, matlib)
```

Spectral density function

a) Below we will generalise an ARMA(p,q) process and its the corresponding spectral density function $S_X(f)$.

$$X_t = \sum_{i=1}^{p} \phi_{i,p} X_{t-p} + \epsilon_t - \sum_{j=1}^{q} \theta_{j,q} \epsilon_{t-q} \quad S_X(f) = \sigma_\epsilon^2 \left| \frac{1 - \theta_{1,q} e^{-2\pi i f} - \theta_{q,q} e^{-2\pi i f q}}{1 - \phi_{1,p} e^{-2\pi i f} - \phi_{p,p} e^{-2\pi i f p}} \right|^2$$

```
S_ARMA <- function(f,phis,thetas,sigma2){
  p <- length(phis) #find orders of process p,q
  q <- length(thetas)
  est <- c()
  for(i in 1:length(f)){
    # num is numerator, den is denominator of fraction
    if(q == 0){num <- 1
    } else { #if we have AR(p) process, num is 1
      num <- abs(1 - sum(to_vec(for(j in 1:q) thetas[j]*exp(-2*pi*f[i]*j*1i) )))**2
      }
    if(p == 0){den <- 1
    } else { #if we have MA(q) process, den is 1
      den <- abs(1 - sum(to_vec(for(j in 1:p) phis[j]*exp(-2*pi*f[i]*j*1i) )))**2
      } # SDF = sigma2 * numerator/denominator
    est <- c(est, sigma2*num/den)
    }
  est
}
```

b) We now will create a function that simulates N values from an ARMA(2,2) process, with a burn-in method, and assume that the white-noise is Gaussian.

```r
ARMA22_sim <- function(phis,thetas,sigma2,N){
  X <- c(0,0) # initialise, x_1 = x_2 = 0
  # %*% applies dot product to vectors which shortens code
  norms <- rnorm(3*(98+N), 0, sigma2) #simulate all normals outside for loop
  for(j in 3:(100+N)){
  x <- X[(j-1):(j-2)] %*% phis - (c(-1,thetas) %*% norms[(3*j-8):(3*j-6)])
  X <- c(X, x) # calculate and append next value of X_t
  }
  X[101:(100+N)] # return last N values
}
```

c) Define the periodogram $\hat{S^p(f)}$ and the direct spectral estimator $\hat{S^d(f)}$ as follows (Cohen 2021),

$$\hat{S^p(f)} = \frac{1}{N}|\sum_{t=1}^{N} X_t e^{-i2\pi f t}|^2 \quad \hat{S^d(f)} = |\sum_{t=1}^{N} h_t X_t e^{-i2\pi f t}|^2$$

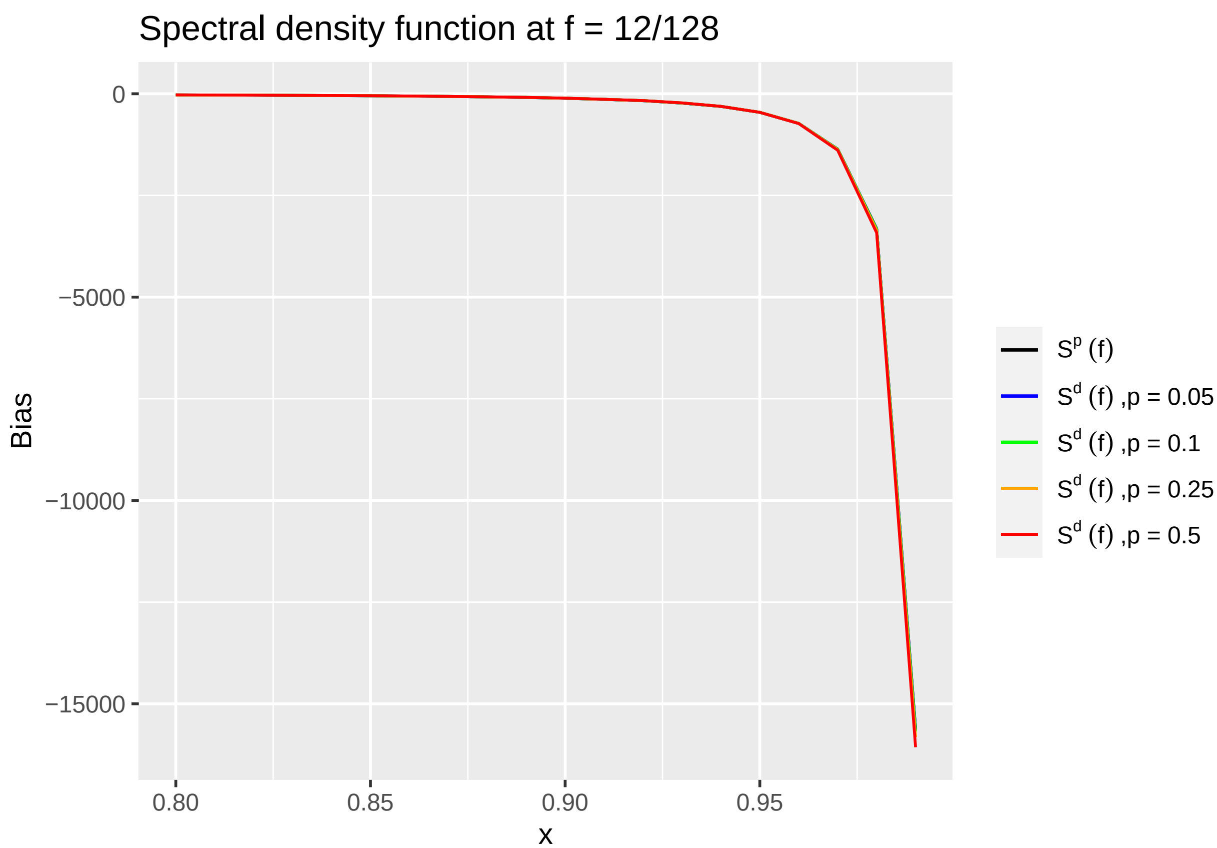
```r
periodogram <- function(X){ abs(fft(X, inverse = FALSE))**2/length(X) }

cos_taper <- function(p, N, t){ #computing unnormalised cosine taper
  if(1 <= t & t <= floor(p*N)/2){
    0.5*(1-cos((2*pi*t)/(floor(p*N)+1)) )
  } else if(floor(p*N)/2 < t & t < N+1-floor(p*N)/2)
    1 else{
    0.5*(1-cos((2*pi*(N+1-t))/(floor(p*N)+1)) )
    }
}
h <- function(p,N){
  #create vector of unnormalise cosine taper values
  h <- to_vec(for(i in 1:N) cos_taper(p,N,i))
  #normalise so that sum of square of values is 1
  h/(sum(h**2)**0.5)
}

#compute periodogram of component wise product of time
# series and cosine taper
direct<- function(X,p){length(X)*periodogram(X*h(p,length(X)))}
```

d) We will now simulate from ARMA(2,2) processes, each with complex conjugate roots of the characteristic polynomial. The time series will show pseudo-cyclical behaviour at $f = \pm\frac{12}{128}$. After taking 10000 simulations for each series, we will calculate the sample bias for periodogram and direct spectral estimate for different cosine tapers.

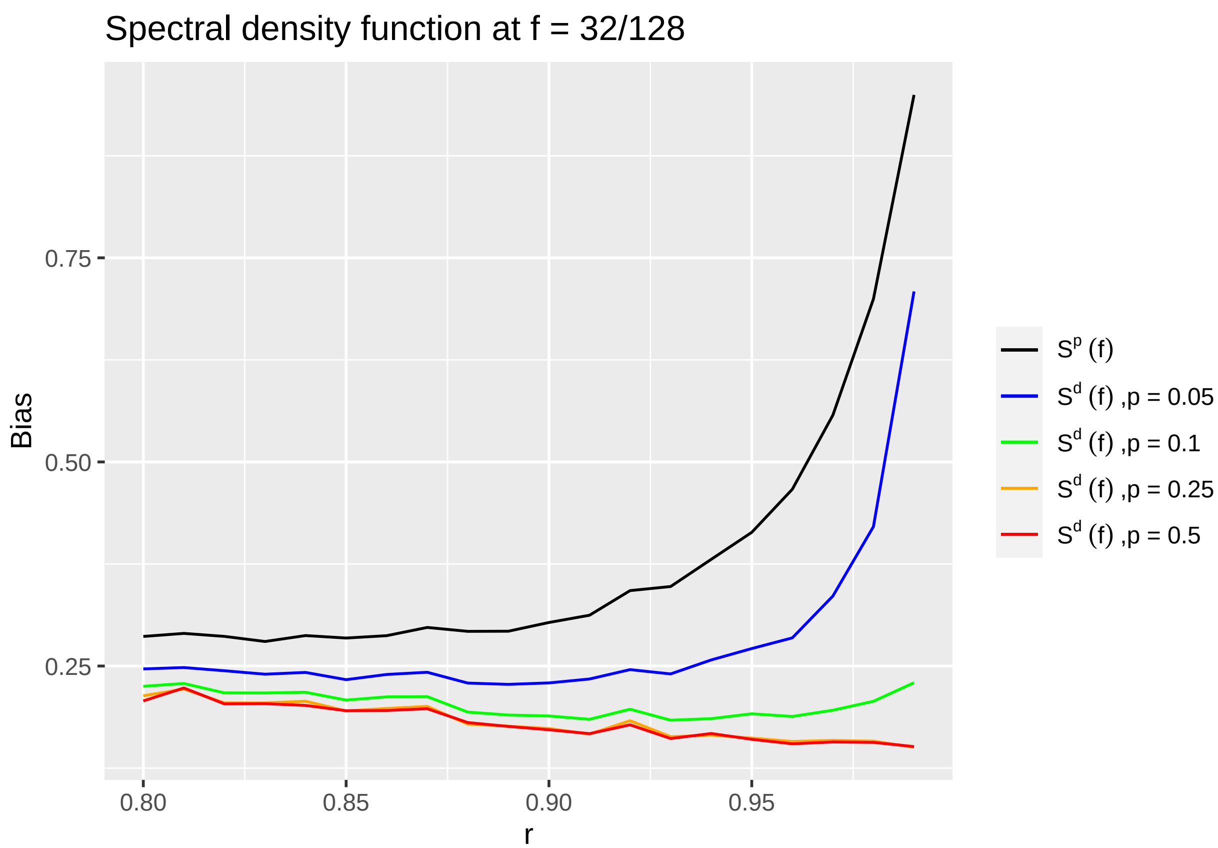
```r
N= 128 # length of time series
r = seq(0.8,0.99,0.01)
freq = c(13,33,61) #indexes of frequencies 12/128,32/128,60/128
freq_val = c(12/128, 32/128, 60/128)
thetas = c(-0.5,-0.2)
p = c(0.05,0.1,0.25,0.5)

f_1 = matrix(nrow = 5, ncol = 20)
f_2 = matrix(nrow = 5, ncol = 20)
f_3 = matrix(nrow = 5, ncol = 20)
```

```r
f = list(f_1,f_2,f_3)#each matrix represents a frequency, inside in each matrix,
#rows are estimator type, columns are different r values

for(j in 1:20){#iterates through r values
  vals <- data.frame(matrix(nrow = 0, ncol = 3))
  vals <- list(vals,vals,vals,vals,vals) #initialize sample bias matrices
  sims <- Reduce(rbind, lapply(1:10000, function(z){ARMA22_sim(c(2*r[j]*cos(2*pi*12/128),
                                                              -r[j]**2),thetas,1,N)}))
  #store simulations from ARMA process
  for(m in 1:5){
      if(m == 1) {vals[[1]] <- Reduce(rbind, lapply(1:10000, function(z){periodogram(
        sims[z,])[freq]}))
      #compute sample mean for periodogram
      } else {vals[[m]] <- Reduce(rbind, lapply(1:10000, function(z){direct(sims[z,],
                                                                 p[m-1])[freq]})) }
    #compute sample mean for direct spectral estimator
    for(l in 1:3){# append sample means
    f[[l]][m,j] <- colMeans(vals[[m]])[[l]] - S_ARMA(freq_val[l],
                                 c(2*r[j]*cos(2*pi*12/128), -r[j]**2),thetas, 1)}
  } #append sample bias
}
```

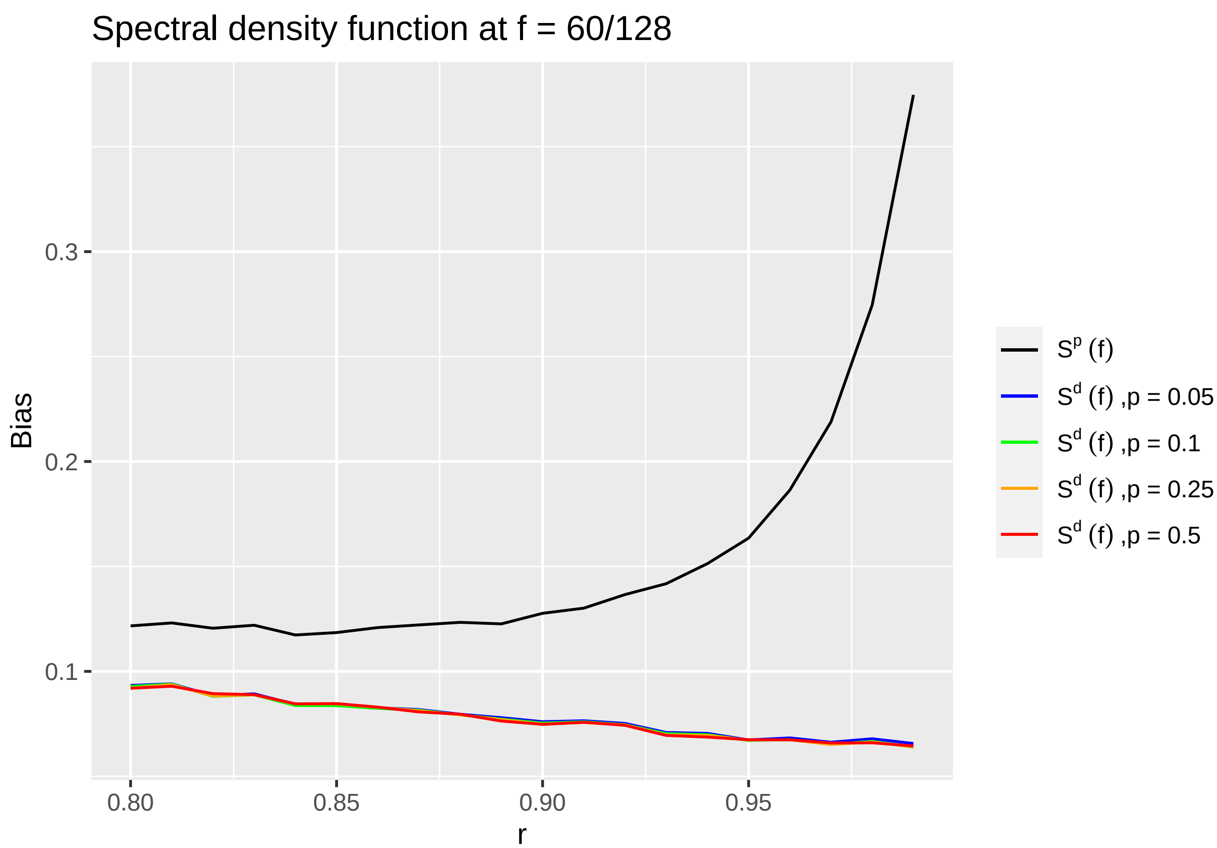
```r
data1 = data.frame(x=r, est1 = f[[1]][1,], est2 = f[[1]][2,], est3 = f[[1]][3,],
                   est4 = f[[1]][4,], est5 = f[[1]][5,])

label=list(expression(S^p~(f)),expression(S^d~(f) ~ ",p = 0.05"),
           expression(S^d~(f) ~ ",p = 0.1"), expression(S^d~(f) ~ ",p = 0.25"),
           expression(S^d~(f) ~ ",p = 0.5"))

ggplot(data1) + geom_line(aes(x,est1,colour="est1"))+geom_line(aes(x,est2,
colour="est2")) + geom_line(aes(x,est3,colour="est3")) + geom_line(aes(x,est4,colour="est4"))+
  geom_line(aes(x,est5,colour="est5")) + labs(y = "Bias",
  title="Spectral density function at f = 12/128") + scale_colour_manual("", values=c("est1"=
    "black","est2"="blue","est3"="green","est4" ="orange","est5"="red"), labels = label)
```




```
data2 = data.frame(x=r, est1 = f[[2]][1,], est2 = f[[2]][2,], est3 = f[[2]][3,],
                   est4 = f[[2]][4,], est5 = f[[2]][5,])

label=list(expression(S^p~(f)),expression(S^d~(f) ~ ",p = 0.05"),
expression(S^d~(f) ~ ",p = 0.1"), expression(S^d~(f) ~ ",p = 0.25"),
expression(S^d~(f) ~ ",p = 0.5"))

ggplot(data2) + geom_line(aes(x,est1,colour="est1"))+geom_line(aes(x,est2,colour="est2")) +
  geom_line(aes(x,est3,colour="est3")) + geom_line(aes(x,est4,colour="est4")) +
  geom_line(aes(x,est5,colour="est5")) + labs(x = "r",y = "Bias",
title="Spectral density function at f = 32/128") + scale_colour_manual("",
values=c("est1"="black","est2"="blue","est3"="green","est4" ="orange","est5"="red"),
labels = label)
```



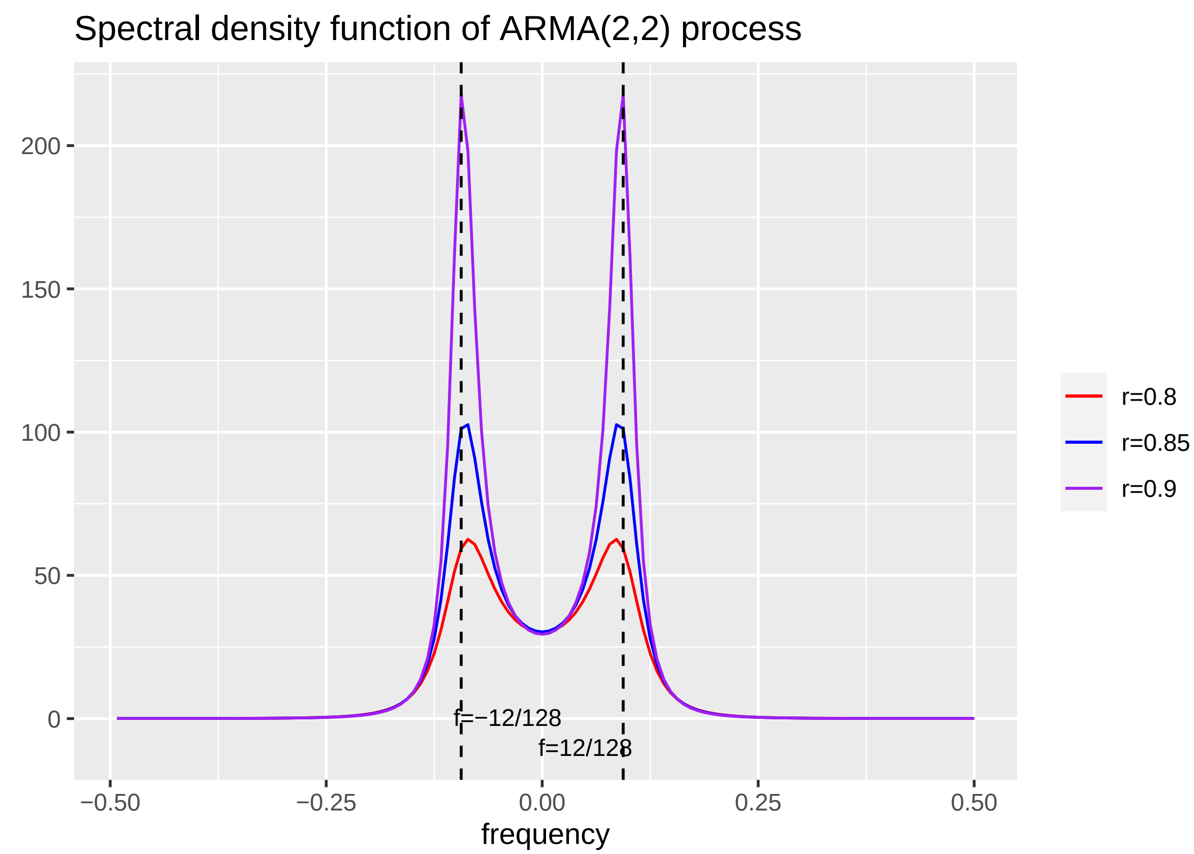

```
data3 = data.frame(x=r, est1 = f[[3]][1,], est2 = f[[3]][2,], est3 = f[[3]][3,],
                   est4 = f[[3]][4,], est5 = f[[3]][5,])

label=list(expression(S^p~(f)),expression(S^d~(f) ~ ",p = 0.05"),
           expression(S^d~(f) ~ ",p = 0.1"), expression(S^d~(f) ~ ",p = 0.25"),
           expression(S^d~(f) ~ ",p = 0.5"))

ggplot(data3) + geom_line(aes(x,est1,colour="est1"))+geom_line(aes(x,est2,colour="est2")) +
  geom_line(aes(x,est3,colour="est3")) + geom_line(aes(x,est4,colour="est4"))+
  geom_line(aes(x,est5,colour="est5")) + labs(x = "r",y = "Bias", title=
"Spectral density function at f = 60/128") + scale_colour_manual("", values=c("est1"=
"black","est2"="blue","est3"="green","est4" ="orange","est5"="red"), labels = label)
```



e) The periodogram is equivalent to a rectangular taper, but as p increases, the p x 100% cosine taper closer represents a delta function which means that $E(\hat{S}^p(f)) \to S(f)$ faster (as $N \to \infty$). This is evident in the plots above where the sample bias reduces as there is further tapering, or p increases.(Cohen 2021)

For f= $\frac{12}{128}$, the estimators for the spectral density function generally have a much larger bias as opposed to the frequencies $\frac{30}{128}$, $\frac{60}{128}$. Note only does the spectral density function reach a peak at this frequency, but if the frequency is fixed and $r \to 1$, the spectral density function tends to ∞.

```
r = c(0.8,0.85,0.9)
freq_range <- (1:128)/128 - 0.5
sdf_mat <- matrix(nrow = 3, ncol = 128)
for(j in 1:3){
  sdf_mat[j,] = S_ARMA(freq_range,c(2*r[j]*cos(2*pi*12/128), -r[j]**2),thetas,1)
}
sdf_mat <- data.frame(x=freq_range, y1 = sdf_mat[1,],y2 = sdf_mat[2,],y3 = sdf_mat[3,])

ggplot(sdf_mat) + geom_line(aes(x=x,y=y1,colour="est1"))+geom_line(aes(x=x,y=y2,
colour="est2")) + geom_line(aes(x=x,y=y3,colour="est3")) + labs(x = "frequency",
y = "", title="Spectral density function of ARMA(2,2) process") +
scale_colour_manual("", values=c("est1"="red","est2"="blue","est3"="purple"),
labels = c("r=0.8","r=0.85","r=0.9")) + geom_vline(xintercept = c(-12/128, 12/128),
colour = "black", linetype ="dashed") + annotate(geom= "text",
label= list('paste(f, "=12/128")'), parse = TRUE, x = 0.05, y = -10, size = 3) +
annotate(geom= "text",label= list('paste(f, "=-12/128")'), parse = TRUE,
x = -0.04, y = 0.5, size = 3)
```



2)

```
estimator_min <- sdf_mat[128,2:4]
estimator_max <- to_vec(for(i in 2:4) max(sdf_mat[,i]))
dynamic_range <- round(10*log(estimator_max/estimator_min),3)
d1 <- dynamic_range[[1]]
d2 <- dynamic_range[[2]]
d3 <- dynamic_range[[3]]
```

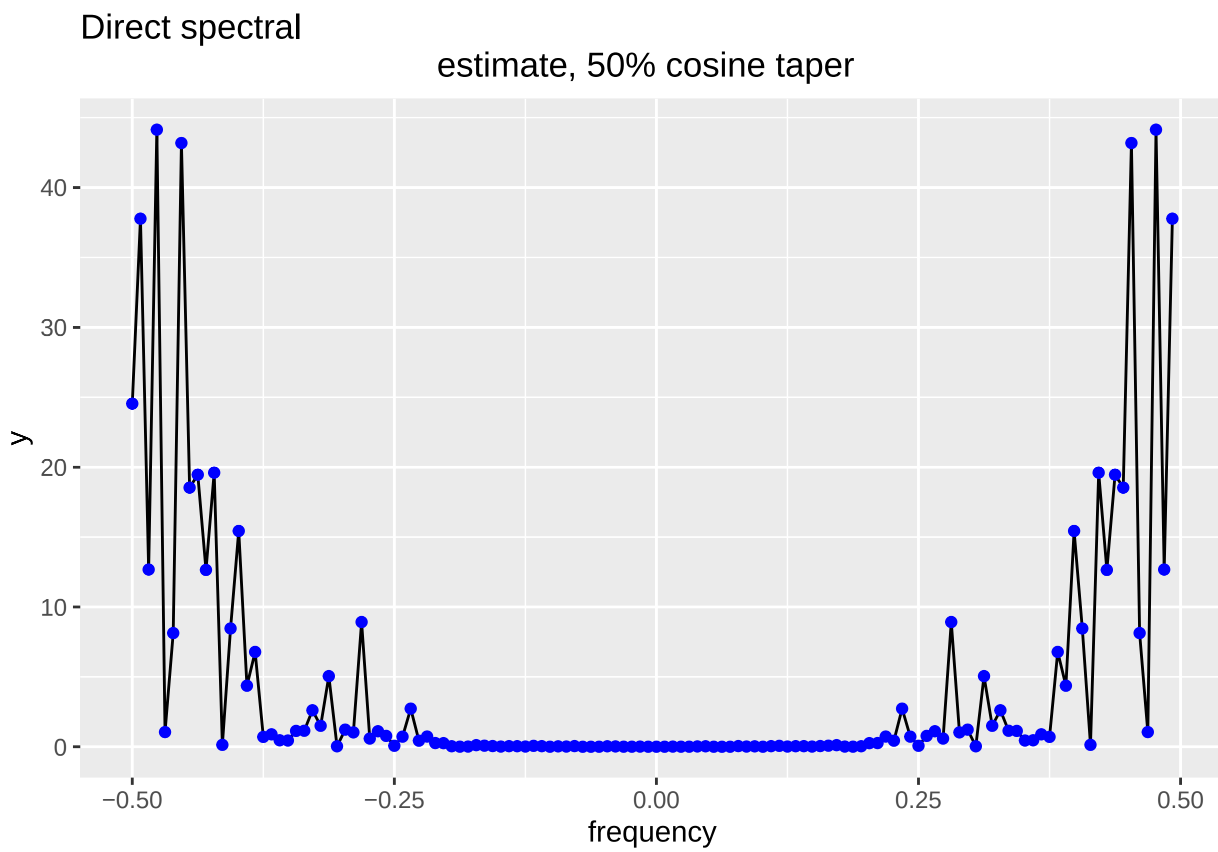
The dynamic range is defined as $10\log(\frac{max_f S(f)}{min_f S(f)})$ (Cohen 2021). For r = 0.8,0.85,0.9 the corresponding dynamic ranges are 70.27, 76.302,84.884 dB. The larger r is, the higher the dynamic range and sidelobe leakage is which increases the bias at frequencies where the spectral density function is small.

Parametric spectral estimation

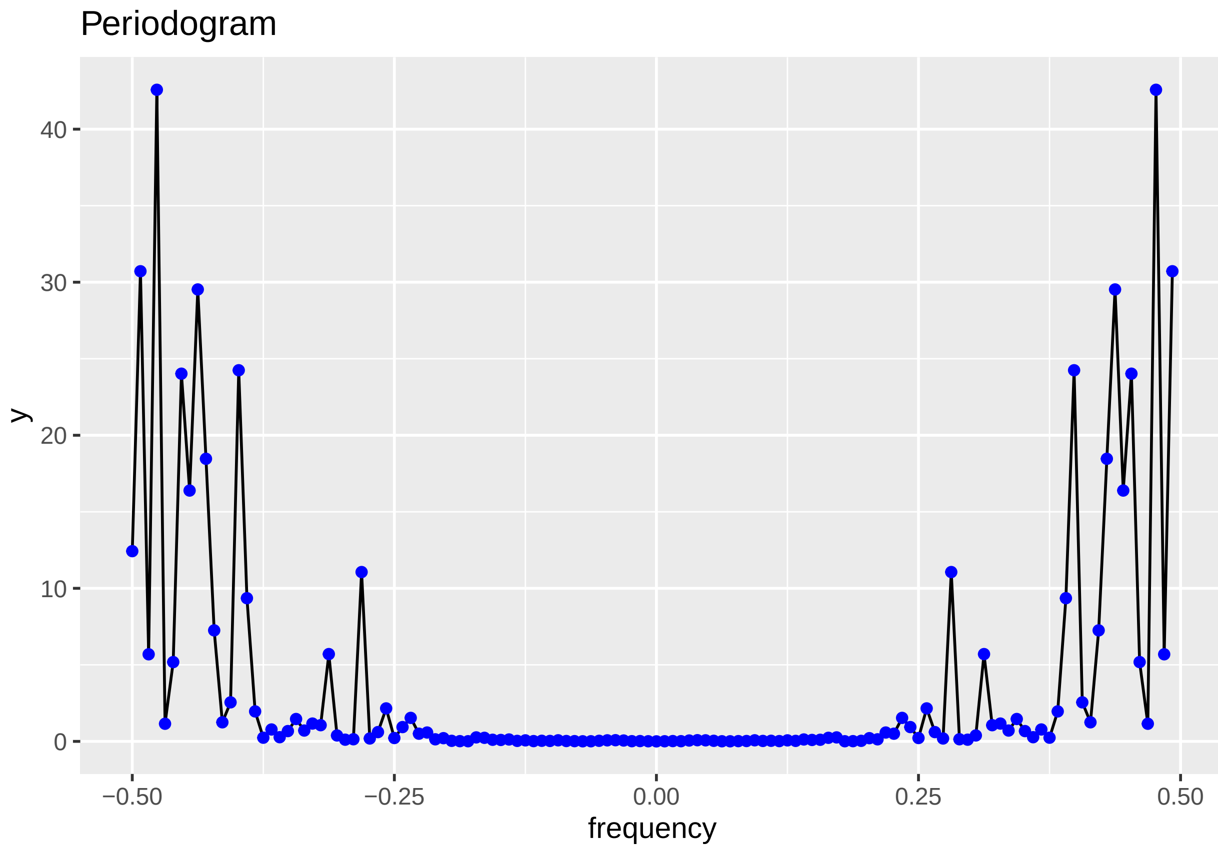
a) We will load the time series, plotting the periodogram and direct spectral estimator with a 50% cosine taper.

```
N=128
f <- seq(0,1-1/N, 1/N) - 0.5
series <-  read_excel("176.xlsx")[[1]] #load time series and extract

p1 <- data.frame(x = f, y = fftshift(direct(series,0.5)))
p2 <- data.frame(x = f, y = fftshift(periodogram(series)))
```

```r
#create dataframe containing frequencies and corresponding estimators

ggplot(data=p1, aes(x=x, y=y)) +
  geom_line(color="black")+
  geom_point(colour = "blue") + labs(x = "frequency", title = "Direct spectral
                                      estimate, 50% cosine taper")
```




```r
ggplot(data=p2, aes(x=x, y=y)) +
  geom_line(color="black")+
  geom_point(colour = "blue") + labs(x = "frequency", title = "Periodogram")
```



b) Below are functions to fit a time series to an AR(p) processes, using the methods: untapered Yule-Walker, Yule-Walker with a 50% cosine taper and approximate maxmimum likelihood.

```
yw_untapered <- function(x,p){
  #take vector of time series as input and fit to AR(P)
  n = length(x)
  s_est <- to_vec( for(j in 1:p) t(x[1:(n-j)]) %*% x[(j+1):n] )/n
  #covariance estimators
  s_0 <- sum(x**2)/n #variance estimator
  if(p==1){alpha <- matrix(s_0)
  } else alpha <- toeplitz( c(s_0, s_est[1:(p-1)]) )
  phi <- solve(alpha, s_est) #solve yule-walker eqns
  sigma <- s_0 - t(phi) %*% s_est
  list(phi, sigma) #output parameter estimators
}
```

```
#simply apply previous function to tapered time series
yw_tapered <- function(x,p){
  yw_untapered(x*h(0.5,length(x)),p)
}
```

```
yw_mle <- function(x,p){
  n = length(x)
  X = x[(p+1):n]
  F = matrix(nrow = n-p, ncol = p)
  for(i in 1:(n-p)){
```

```
    for(j in 1:p){
      F[i,j] = x[p +i-j]
    }
  }
  phi <- solve(t(F)%*%F, t(F)%*%X)
  sigma <- sum((X - F%*%phi)**2)/(n-p)
  list(phi, sigma)
}
```

c) We will assume our time series is Gaussian and plot its Akaike information criterion (AIC) for various p-values. This is defined as follows

$$AIC = 2p + 2Nln(\hat{\sigma_\epsilon})$$

```
N = length(series)
aic_vals <- matrix(nrow = 20, ncol = 3)
for(i in 1:20){
  aic_vals[i,1] = round(2*i + N*log(yw_untapered(series,i)[[2]]),2)
  aic_vals[i,2] = round(2*i + N*log(yw_tapered(series,i)[[2]]),2)
  aic_vals[i,3] = round(2*i + N*log(yw_mle(series,i)[[2]]),2)
}
aic_vals <- cbind(1:20,aic_vals)
kable(aic_vals, col.names =c("p","YW(untapered)","YW(50% cosine taper )", "MLE"))
```

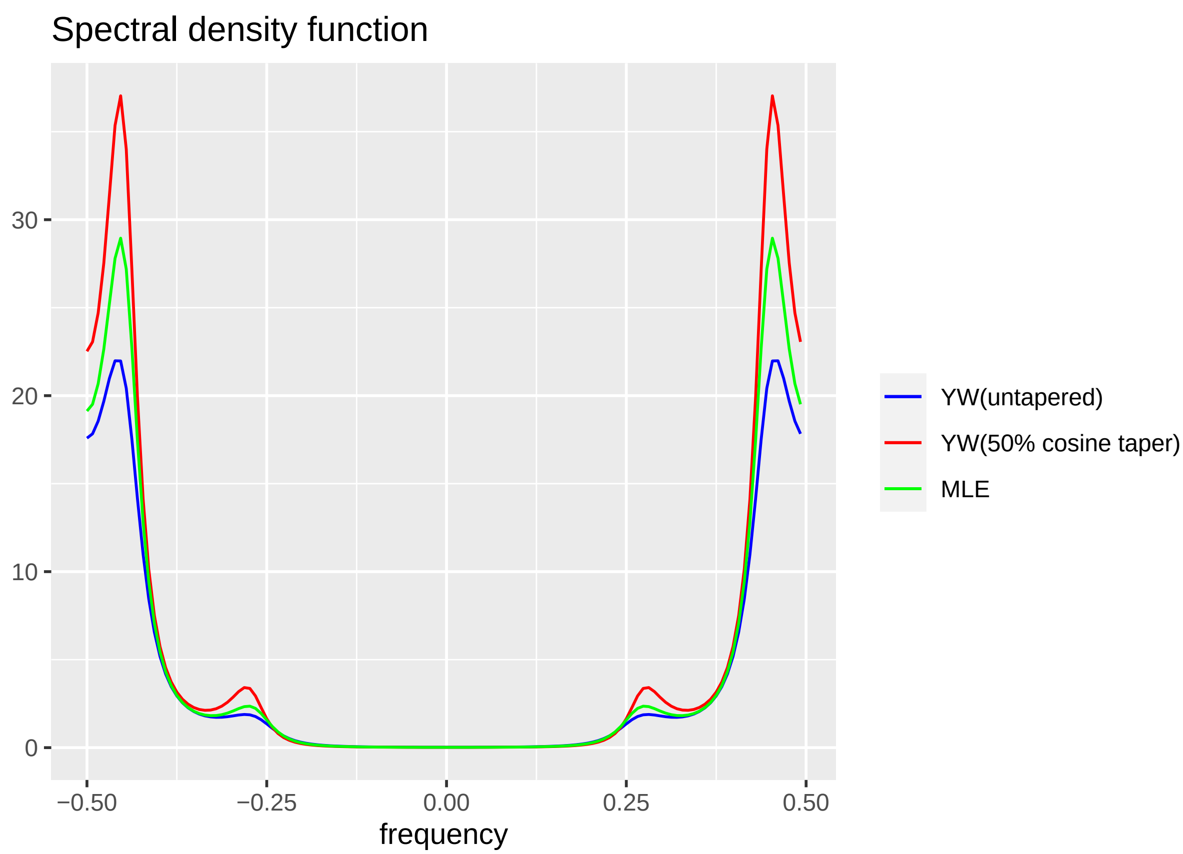
p	YW(untapered)	YW(50% cosine taper)	MLE
1	17.93	-595.41	12.48
2	-16.84	-636.52	-26.77
3	-36.96	-667.81	-50.24
4	-53.83	-701.06	-74.59
5	-52.21	-700.10	-72.77
6	-50.36	-698.14	-69.90
7	-49.80	-696.69	-67.73
8	-48.33	-694.98	-66.04
9	-46.34	-694.70	-63.60
10	-44.36	-695.24	-60.52
11	-43.36	-693.89	-58.74
12	-42.02	-692.42	-55.67
13	-44.87	-690.83	-55.67
14	-42.93	-688.97	-52.85
15	-43.08	-689.70	-53.71
16	-43.00	-689.17	-54.94
17	-41.60	-687.18	-51.98
18	-48.33	-691.73	-57.42
19	-46.68	-689.74	-55.20
20	-45.84	-687.76	-52.74

d) For each method, we get the lowest AIC score, and best fitting model, when the order of the autoregressive process is 4. The AIC scores respectively are -53.83, -701.06 and -74.59 respectively. Below we display the parameters $\hat{\phi_{1,4}}, \hat{\phi_{2,4}}, \hat{\phi_{3,4}}, \hat{\phi_{4,4}}, \hat{\sigma_\epsilon}^2$.

```r
p1 <- c(yw_untapered(series,4)[[1]],yw_untapered(series,4)[[2]])
p2 <- c(yw_tapered(series,4)[[1]],yw_untapered(series,4)[[2]])
p3 <- c(yw_mle(series,4)[[1]],yw_untapered(series,4)[[2]])
parameters <- round(cbind(p1,p2,p3),3)
rownames(parameters) <- c("$\\phi_{1,4}$","$\\phi_{2,4}$","$\\phi_{3,4}$",
                          "$\\phi_{4,4}$","$\\sigma_{\\epsilon}^2$")
kable(parameters, col.names =c("YW(untapered)","YW(50% cosine taper)", "MLE"))
```

	YW(untapered)	YW(50% cosine taper)	MLE
$\phi_{1,4}$	-1.619	-1.813	-1.705
$\phi_{2,4}$	-1.379	-1.740	-1.536
$\phi_{3,4}$	-0.943	-1.253	-1.087
$\phi_{4,4}$	-0.370	-0.491	-0.436
σ_{ϵ}^2	0.617	0.617	0.617

```r
N = length(series)
f <- seq(0,1-1/N, 1/N) - 0.5
sdf <- data.frame(x = f, y1 = S_ARMA(f,p1[1:4],c(),p1[5]),y2 = S_ARMA(f,p2[1:4],
                                   c(),p2[5]),y3 = S_ARMA(f,p3[1:4],c(),p3[5]) )

ggplot(sdf) + geom_line(aes(x,y1,colour="y1"))+geom_line(aes(x,y2,colour="y2")) +
  geom_line(aes(x,y3,colour="y3")) + labs(title="Spectral density function") +
scale_colour_manual("", values=c("y1"="blue","y2"="red","y3"="green"), labels =
c("YW(untapered)","YW(50% cosine taper)", "MLE")) + labs(x = "frequency", y = "")
```



Forecasting

a) To forecast a from an AR(p) process l steps ahead of a given time t, where we use the formula. We will assume p = 4 as this model had the lowest AIC.

$$X_t(l) = \sum_{i=1}^{l-1} \phi_i X_t(l-i) + \sum_{i=l}^{p} \phi_i X_{t+l-i}$$

```
aic_val <- c()
for(i in 1:20){
    aic_val<- c(aic_val, 2*i + N*log(yw_mle(series[1:118],i)[[2]]) )
}
p <- which.min(aic_val)

phis <- yw_mle(series[1:118], p)[[1]]
forecast <- c(t(phis)%*%series[118:(118-p+1)]) # set first forecast X_119
for(j in 2:10){ # compute forecasts X_120, .. , X_128
  if(j <= p) { x <- t(phis)%*%c(rev(forecast), series[118:(118-p+j)])
  #forecast depends on some observed X values
  } else { x <- t(phis)%*%rev(forecast)[1:p] }
  #forecast depends only on previous forecasts
  forecast <- c(forecast, x)
}

ar4_forecast <- forecast #store forecasts for AR(4) process

ar_forecast <- round(rbind(series[119:128],ar4_forecast),3)
rownames(ar_forecast) <- c("Actual","Predicted")
kable(ar_forecast, col.names =c("$X_{119}$","$X_{120}$","$X_{121}$","$X_{122}$",
"$X_{123}$","$X_{124}$","$X_{125}$","$X_{126}$","$X_{127}$","$X_{128}$"),
caption = "Predicted vs Actual values, assuming AR(4) process")
```

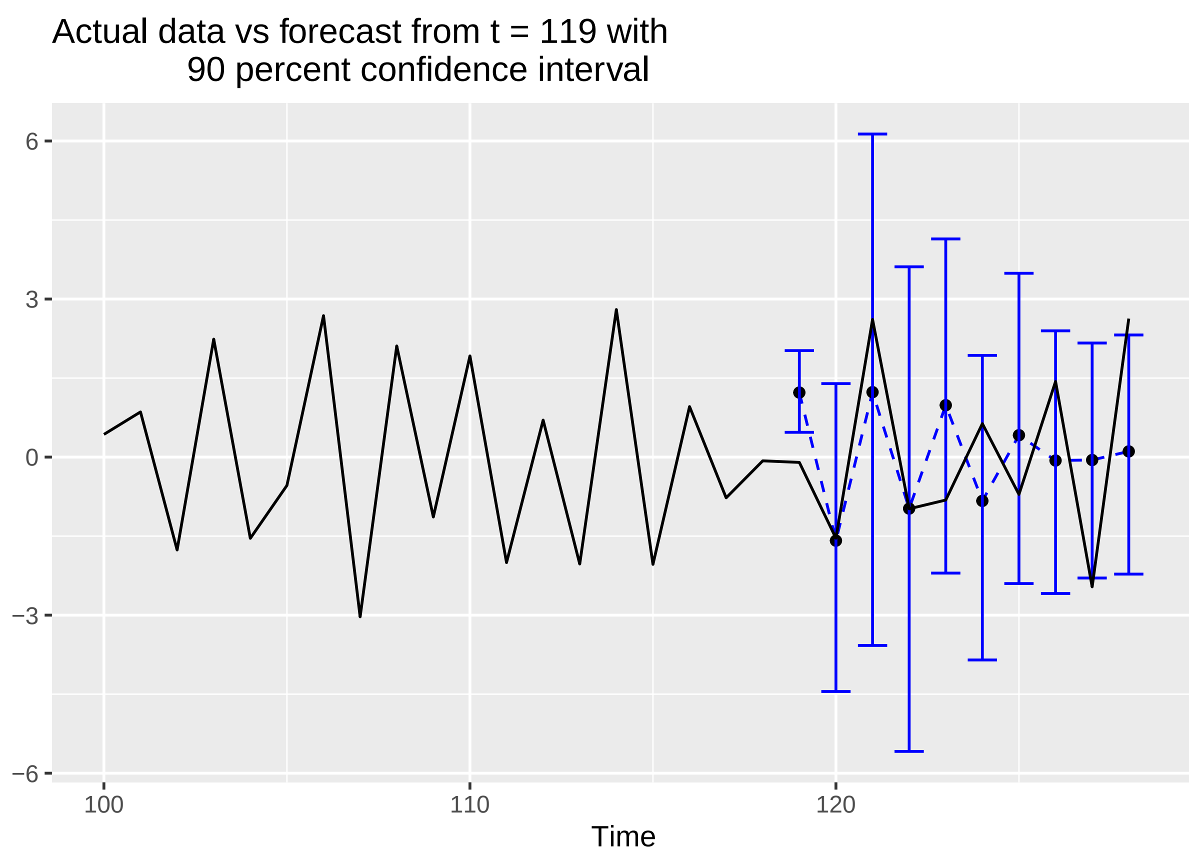
Table 3: Predicted vs Actual values, assuming AR(4) process

	X_{119}	X_{120}	X_{121}	X_{122}	X_{123}	X_{124}	X_{125}	X_{126}	X_{127}	X_{128}
Actual	-0.101	-1.540	2.613	-0.981	-0.815	0.635	-0.709	1.436	-2.463	2.629
Predicted	1.225	-1.586	1.233	-0.977	0.985	-0.831	0.416	-0.065	-0.055	0.108

b) Now we will use point forecasts to design an approximate 90 percent confidence interval with our forecast, we will also simulate the innovation terms $\epsilon_{119}, ..\epsilon_{128}$ rather than setting them equal to 0. In the forecast plot below, the actual values of the time series are black, and the forecast is blue.

```
forecast_set <- data.frame(matrix(nrow = 0, ncol= 10))
# (i,j)th element is forecast j steps ahead of x_118 on i'th simulation


ar4_fit <- yw_mle(series[1:118], 4)
phis <- ar4_fit[[1]]
sigma2 <- ar4_fit[[2]] # extract variance of ar(4) fit

for(k in 1:1000){#1000 simulations
```

```r
  forecast <- c(t(phis)%*%series[118:(115)] + rnorm(1,0,sigma2)) # set first forecast X_119
  for(j in 2:10){ # compute forecasts X_120, .. , X_128
    if(j <= 4) { x <- t(phis)%*%c(rev(forecast), series[118:(114+j)] + rnorm(1,0,sigma2))
    #forecast depends on some observed X values
    } else { x <- t(phis)%*%rev(forecast)[1:4] + + rnorm(1,0,sigma2) }
    #forecast depends only on previous forecasts
    forecast <- c(forecast, x)
  }
  forecast_set <- rbind(forecast_set, forecast)
}

# extract 50th and 950th highest forecast out of 1000
upperpred <- to_vec(for(i in 1:10) quantile(forecast_set[,i], 949/999))
lowerpred <- to_vec(for(i in 1:10) quantile(forecast_set[,i], 49/999))

forecast_plot1 <- data.frame(x1 = 100:128, y4 = series[100:128])
forecast_plot2 <- data.frame(x1 = 119:128, y1 = lowerpred, y2 = upperpred,
                             y3 = ar4_forecast)

ab <- ggplot() + geom_line(data = forecast_plot1, aes(x = x1,y = y4), colour = "black")
ggplot(data = forecast_plot2, aes(x1,y3)) + geom_line(colour = "blue",
linetype = "dashed") + geom_point() + geom_errorbar(aes(ymin=lowerpred, ymax=upperpred),
width=.8, colour = "blue") + geom_line(data = forecast_plot1, aes(x = x1,y = y4),
colour = "black") +  labs(title = "Actual data vs forecast from t = 119 with
              90 percent confidence interval", x = "Time", y="")
```



Sources

1) Cohen, E. (2021) Time Series Analysis. [Lecture Notes] Imperial College London, 10th Decemeber.