TSA Assignment 6

Christian Hilscher - 1570550

4/24/2020

DGP

First I generate the sample where I draw ϵ_t. From this I compute u_t and then finally x_t. Dimensions are chosen such that in the end I have 1000 observations of x_t.

```
# Function for getting x (one sample)
get_x <- function(length, a, beta){
  # Need more epsilons since I have two lags
  epsilon <- rnorm(length+2)
  # Calculating e
  e_tmp <- epsilon - (5/6)*shift(epsilon,1) + (1/6)*shift(epsilon,2)
  e <- e_tmp[3:length(e_tmp)]

  #Calculating u
  u_initial <- e[1]
  u <- u_initial
  for (i in seq(2,ts_length)){
    u_t <- a*u[i-1] + e[i]
    u <- c(u, u_t)
  }

  # Calculating x
  x <- beta_0 + u
  return(x)
}

############################################
# Setting the parameters and calulating
############################################


ts_length <- 1000
sample_size <- 1000
a <- 1
beta_0 <- 1

x_mat <- matrix(NA, sample_size, ts_length)

# Filling up the matrix; each row is sample
for (sample in seq(sample_size)){
  x_mat[sample,] <- get_x(ts_length, a, beta_0)
}
```

Estimating

Having the time series, I calculate the individual t-statistics for each sample and strategy.

```r
ratios <- get_tratios(x_mat, 1, sample_size)
```

Where the (albeit somewhat clunky) function *get_tratios* takes as arguments the matrix containing the 1000 samples, the true value a and the sample size S. All auxillary functions are shown and explained in the appendix.

```r
get_tratios <- function(x_mat, a, sample_size){
  t_ratios1 <- vector()
  t_ratios2 <- vector()
  t_ratios3 <- vector()

  for (sample in seq(sample_size)){

    # Frist getting the coefficients, gamma_e and the errors themselves
    d <- get_coef_se(x=x_mat[sample,])

    # Strategy 1
    t1 <- strategy1(a=1,
                    a1_hat=d[['coef_and_se']][1],
                    a1_hat_se=d[['coef_and_se']][2])
    t_ratios1 <- cbind(t_ratios1, t1)

    # Strategy 2
    t2 <- strategies_lrvar(a=1,
                           a1_hat = d[['coef_and_se']][1],
                           a1_hat_se = d[['coef_and_se']][2],
                           gamma_e = d[['gamma_e']],
                           e = d[['e']],
                           strategy="strategy2")
    t_ratios2 <- cbind(t_ratios2, t2)

    # Strategy 3
    t3 <- strategies_lrvar(a=1,
                           a1_hat = d[['coef_and_se']][1],
                           a1_hat_se = d[['coef_and_se']][2],
                           gamma_e = d[['gamma_e']],
                           e = d[['e']],
                           strategy="strategy3")
    t_ratios3 <- cbind(t_ratios3, t3)
  }

  # Combining results
  out_mat <- matrix(c(t_ratios1,
                      t_ratios2,
                      t_ratios3), nrow=3,
                    ncol=1000, byrow = TRUE)
  return(out_mat)
}
```

From here the matrix *ratios* contains the $t - statistics$ where each row correpsonds to a differnt strategy and each column to another sample. Now I simply look whether each value is above or below the critical value and this way geting the probability to reject H_0.

```r
results <- calculate_P(ratios, (-2.86))

# Function for calculating the probability to reject H_0
calculate_P <- function(ratio_mat, critical_val){
  samples <- dim(ratio_mat)[1]
  p_list <- vector()

  for (sample in seq(samples)){
    values <- ratio_mat[sample,]
    p <- sum((values<(critical_val))/length(values))
    p_list <- cbind(p_list, p)
  }
  return(p_list)
}
```

The corresponding values are then given by:

Table 1: Probability of rejecting H_0

Strategy 1	Strategy 2	Strategy 3
0.959	0.993	0.829

Since the DGP is known and implemented such that $a = 1$ ideally the tests should not reject null hypothesis. This however is not the case with all of the three tests rejecting almost every time. I'd prefer that particular test which rejects the null the least often, making strategy 3 the "best" of the three even though that test is not too good either.

The bad performance of the tests could be partly explained by not really being able to distinguish between the stochastic and deterministic trend. Running an ERS test, it keeps H_0 when using $\alpha = 0.1$. This could be seen as evidence pointing to pretty large difficulties keeping the stochastic and deterministic trend apart since the ERS at least partly solves issue.

Exploring why Strategy 2 and 3 differ

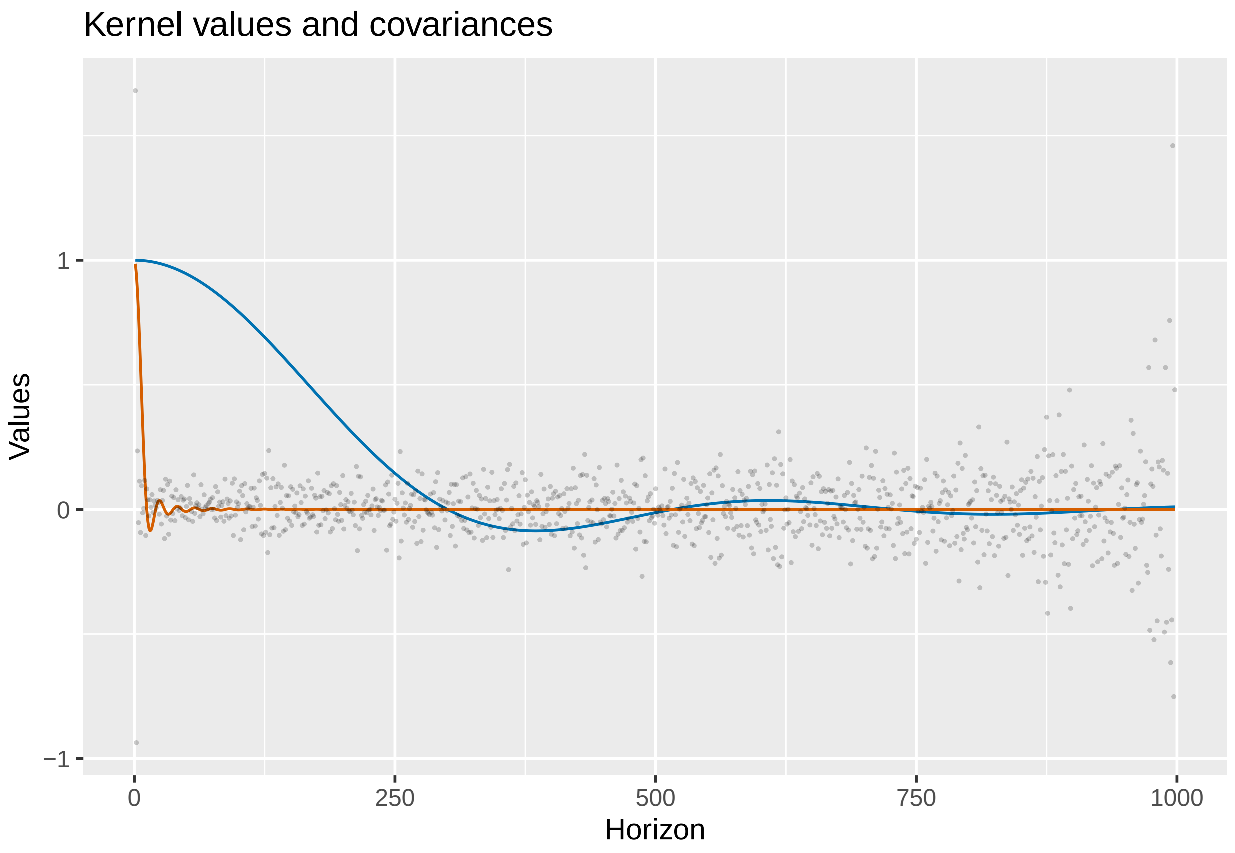
The only way strategy 2 and 3 differ is the q used in the kernel when calculating the long run varaince. This in turn determines how much weight the individual covariances in the sum get. Choosing a random sample from the matrix containing all samples I plot the value of the kernel depending on which strategy is used as well as the covariances. Here the blue line corresponds to strategy 2, the orange line is strategy 3 and the little black dots are the respective autocovariances.

```r
ran <- sample(1:sample_size, 1)
random <- get_coef_se(x_mat[ran,])
res <- random[['e']]
values <- seq(1, 998)

df1 <- plotting(res)

p1 = ggplot(data=df1, aes(x=horizon)) +
   geom_line(aes(y=strategy2), color="#0072B2") +
   geom_line(aes(y=strategy3), color="#D55E00") +
   geom_point(aes(y=cov), alpha=0.2, colour="black", size=0.2) +
   xlab('Horizon') +
   ylab('Values') +
   ggtitle('Kernel values and covariances')
```

```
print(p1)
```



One can see that the kernel used in the more accurate strategy 3 is putting a lot of weight on the first couple of covariances but almost none on the latter ones. In contrast, the kernel from strategy 2 has a large positive weight on the first 250 covariances. From the DGP we know that the error is serially correlated only up to the second lag. That could be one reason why strategy 3 which puts almost no weight on later covariances performs better compared to strategy 2.

Appendix - Auxilary functions

Shifting a vector by certain lag/lead length

```
# Function for shifting the variables back - i.e. get e_{t-1} from e_t
shift <- function(x, lag) {
  n <- length(x)
  xnew <- rep(NA, n)
  if (lag < 0) {
    xnew[1:(n-abs(lag))] <- x[(abs(lag)+1):n]
  } else if (lag > 0) {
    xnew[(lag+1):n] <- x[1:(n-lag)]
  } else {
    xnew <- x
  }
  return(xnew)
}
```

Getting the coefficients

First regressing x_t on x_{t-1}. Then *get_coef_se* returns a dictionary holding the estimate $\hat{a}$ as well as it's standard error $\hat{\sigma}_{\hat{a}}$, the variance of the errors $\hat{\gamma}_e$ and finally the prediction errors $\hat{e}$.

```
get_coef_se <- function(x){
  x_t1 <- shift(x,1)
  x_t1 <- x_t1[2:length(x_t1)]

  y <-  x[2:length(x)]
  ols_frame <- data.frame(y, x_t1)
  modl <- lm(y ~ x_t1, data=ols_frame)
  sum <- summary(modl)

  d <- dict()
  #Adding the coefficient as well as SE
  d[['coef_and_se']] <- sum$coefficients[2,1:2]
  #Now including varaince of errors; needed for strategy 2 and 3
  d[['gamma_e']] <- sum$sigma
  d[['e']] <- sum$residuals
  return(d)
}
```

Calculating k

This function implements the the second equation from slide 76. Given z it calculates the kernel used for the long run variance estimation

```
k <- function(z){
  c <- (6*pi*z)/5
  value <- 3/(c**2) * (sin(c)/(c)-cos(c))
  return(value)
}
```

Calculating covariances

This function is also used when getting the long run varianc. More specifically, it calculates the covaraince of input e and the lag length h

```
covs <- function(x, h){
  x_th <- shift(x,h)[(h+1):length(shift(x,h))]
  x_t <- x[(h+1):length(x)]


  return(cov(x_t, x_th))
}
```

Long run varaince

Here I use the two functions defined above to implement the formula from slide 76. I assumed $k = 2$ since in the regression I estimate a constant as well as $\hat{a}$. Since I only have a univariate time-series the covariances are numbers and that's why I multiply them by 2. Furthermore, depending on which strategy is given as input it uses a different q.

```
get_omega <- function(e, strategy){

  gamma_0 <- var(e)
```

```r
  T <- length(e)
  sum_T <- length(e) - 2

  if (strategy == "strategy2"){
    q <- floor(T**(4/5))
  } else {
    q <- floor(T**(1/3))
  }

  sum_overh <- 0
  for (h in seq(sum_T)){
    z <- (h/(q+1))
    value <-  k(z)*covs(e,h)*2
    sum_overh <- sum_overh + value
  }

  omega_hat <- (T/(T-2))*(gamma_0 + sum_overh)
  return(omega_hat)
}
```

Calculating t-ratios

Depending on the strategy, the tratios are computed differently. Strategy 1 is pretty straight forward with taking as inputs the true value a, the estimated $\hat{a}$ as well as the standard error $\hat{\sigma}_{\hat{a}}$ which are obtained from the regression of x_t on x_{t-1}.

```r
strategy1 <- function(a, a1_hat, a1_hat_se){
  t_ratio <- (a1_hat-a)/a1_hat_se
  return(t_ratio)
}
```

Strategy 2 and 3 use the following function

```r
strategies_lrvar <- function(a, a1_hat, a1_hat_se, gamma_e, e, strategy){

  T <- length(e)
  omega_hat <- get_omega(e, strategy)

  correction <- sqrt(gamma_e/omega_hat)
  secondterm <- ((omega_hat-gamma_e)/(2*sqrt(omega_hat))) * ((T*a1_hat_se)/(gamma_e))

  t_ratio <- correction * (a1_hat-a)/a1_hat_se - secondterm
  return(t_ratio)
}
```

Plotting

Finally, the function used for getting the dataframe to plot the kernel values and covariances is given by

```r
plotting <- function(e){

   T <- length(e) + 1
   sum_T <- length(e) - 1

   horizon <- seq(1, sum_T)
   df = data.frame('horizon' = horizon)
```

```
  for (i in cbind('strategy2', 'strategy3')){
    if (i == "strategy2"){
      q <- floor(T**(4/5))
    } else {
      q <- floor(T**(1/3))
    }

    cov_list <- vector()
    kernel_list <- vector()
    sum_overh <- 0
    for (h in seq(sum_T)){
      z <- (h/(q+1))
      value <-  k(z)
      kernel_list <- rbind(kernel_list, value)

      cov <- covs(e, h-1)
      cov_list <- rbind(cov_list, cov)
    }
    df[i] <- kernel_list
  }
  df['cov'] <- cov_list
  return(df)
}
```