AR model - OLS estimator

Properties of the OLS estimator in an AR context

What will we learn in this empirical exercise ?

1. How to compute the autocorrelation values.
2. Show the bias of the OLS estimator.
3. Show the consistency of the OLS estimator.

Theory behind practice

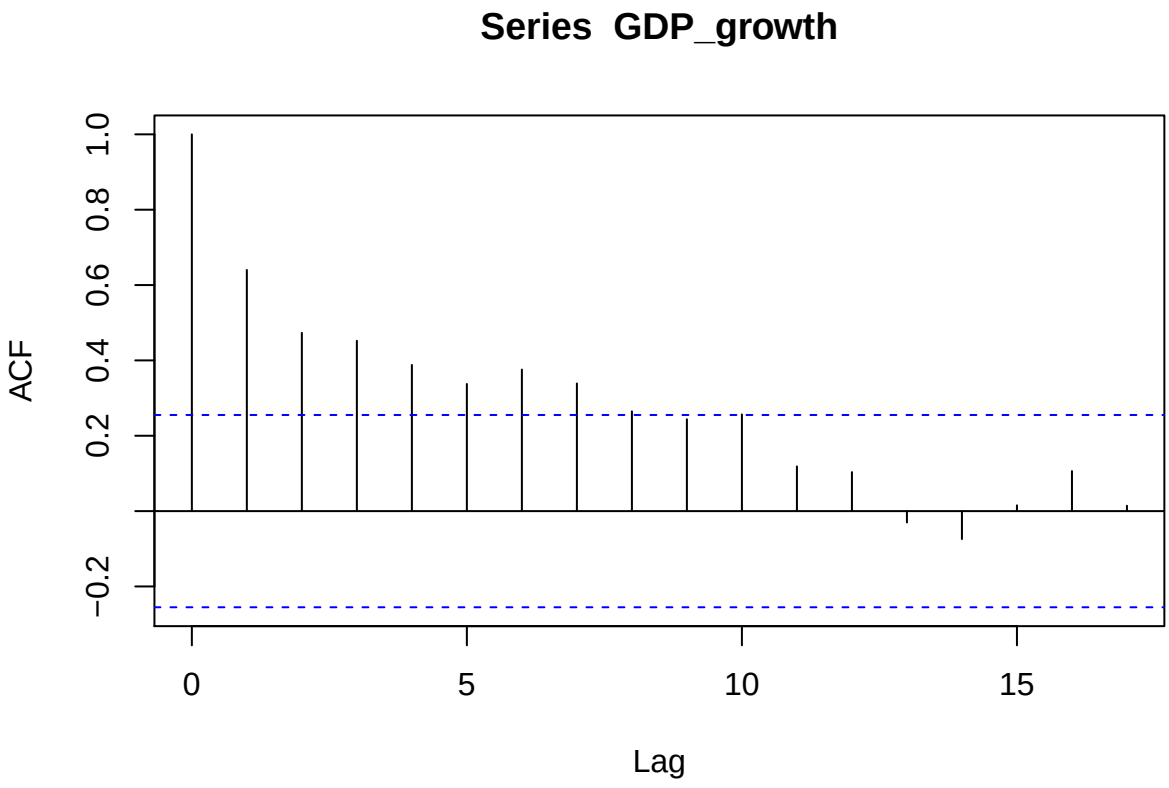
Let us consider an AR(1) model given by $y_t = \beta_1 + \beta_2 y_{t-1} + \epsilon_t$. The model can be reframed as a standard linear regression with one explanatory variable, i.e. $y_t = \beta_1 + \beta_2 x_t + \epsilon_t$ in which $x_t = y_{t-1}$. Unfortunately, the strict exogeneity assumption, i.e. $E(\epsilon_t|x) = 0 \ \ \forall t \in [1, T]$, does not hold since it implies that $E(\epsilon_t y_t) = 0$ while the AR(1) model leads to $E(\epsilon_t y_t) > 0$. The OLS estimator is thus biased. However, we shall show that if the error terms are i.i.d.(0,σ^2) and $|\beta_2| < 1$, then the OLS estimator remains consistent.

In this exercise, we first load the yearly GDP growth rate and compute its autocorrelation values. Then, we estimate an AR(1) model and check if the residuals still exhibits autocorrelation. Using simulated series, we show that the OLS estimator is biased but consistent.

From theory to practice

We first load the yearly GDP growth rate and check its autocorrelation values.

```
url = "https://raw.githubusercontent.com/adufays/GDP_expectancy/main/france-gdp-growth-rate.csv"
data = read.csv(url, sep=",")
GDP_growth = data[,2]
acf(GDP_growth)
autocorr = acf(GDP_growth)
```




```
(autocorr)
```

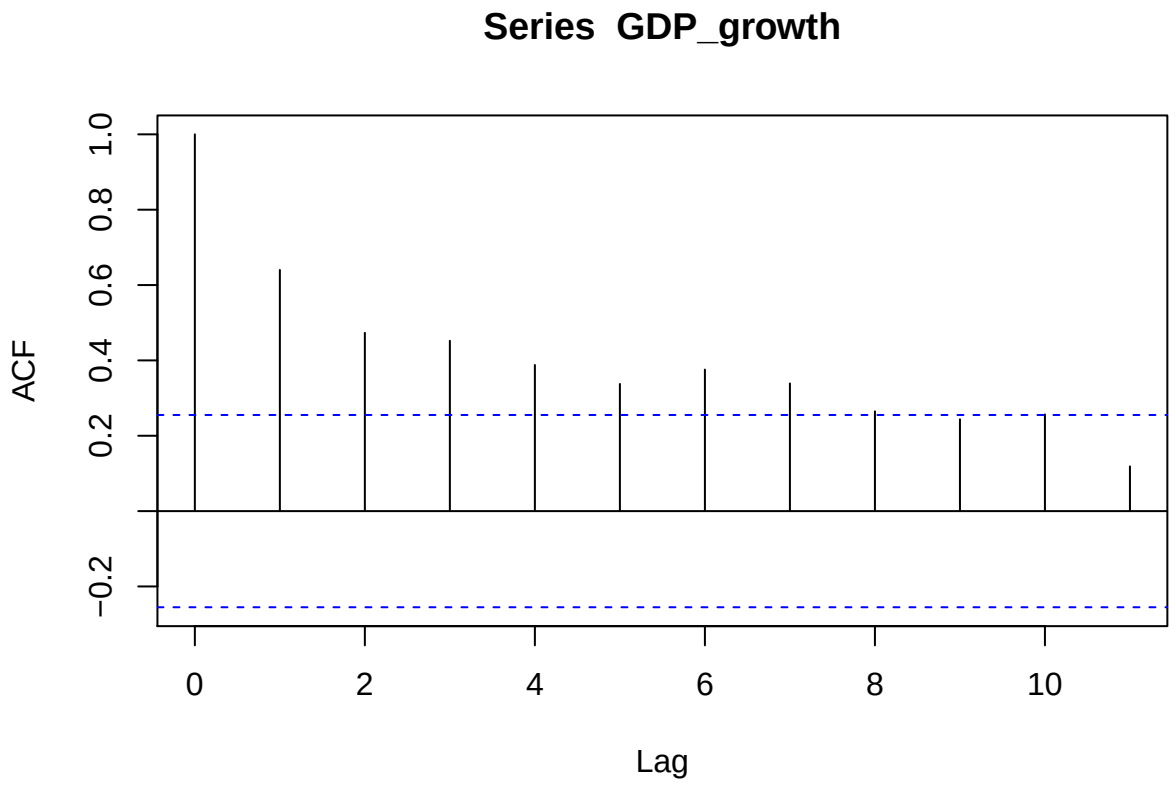
```
##
## Autocorrelations of series 'GDP_growth', by lag
##
##      0      1      2      3      4      5      6      7      8      9     10
##  1.000  0.640  0.473  0.452  0.388  0.338  0.376  0.339  0.265  0.244  0.257
##     11     12     13     14     15     16     17
##  0.119  0.104 -0.030 -0.074  0.015  0.106  0.014
```

We observe the presence of autocorrelation and that the 1-lag autocorrelation is the highest autocorrelation value. Using the Box-Pierce statistical test, we formally verify the presence of autocorrelation up to 10 lags.

```
Box.test(GDP_growth,lag = 10) ## R function
```

```
##
##  Box-Pierce test
##
## data:  GDP_growth
## X-squared = 91.673, df = 10, p-value = 2.442e-15
```

```
## Performing manually the same test
max_lag = 11
acf_val = acf(GDP_growth,lag.max = max_lag)
```




```
val = acf_val$acf
T = length(GDP_growth)
T*sum(val[2:max_lag]^2)
```

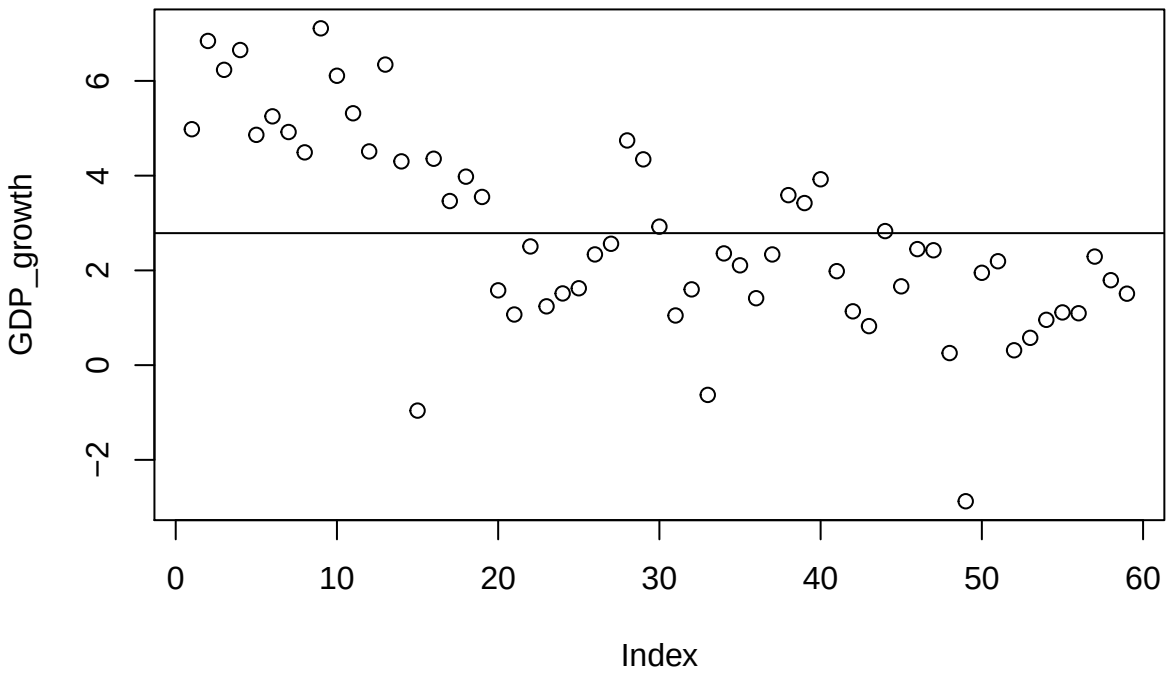
```
## [1] 91.67323
```

```
qchisq(0.95,max_lag-1)
```

```
## [1] 18.30704
```

We clearly reject the Null hypothesis which assumes no autocorrelation up to 10 lags. To further confirm the presence of autocorrelation, we plot the series and its empirical average. We see some persistence in the series meaning that when a value is above (below) the average, it is likely that the next value remains above (below) the average as well.

```
plot(GDP_growth)
abline(a=mean(GDP_growth),b=0)
```



We now estimate an AR(1) model using the OLS estimates. In the next empirical exercise, we will see that the autocorrelation values of an AR(1) model geometrically decrease. So,the autocorrelation plot of the AR(1) model is similar to the autocorrelation graphic that we observed for the GDP growth rate.

```
X = array(1,dim=c(T-1,2))
X[,2] = GDP_growth[1:(T-1)]
y = GDP_growth[2:T]
beta_ols = solve(t(X)%*%X)%*%t(X)%*%y
residuals = y-X%*%beta_ols
(beta_ols)
```

```
##           [,1]
## [1,] 0.9383131
## [2,] 0.6445963
```

As expected, we first observe that the AR estimate, which amounts to 0.645, is almost identical as the 1-lag autocorrelation. We now test for autocorrelation in the residuals.

```
Box.test(residuals,lag = 10) ## R function
```

```
##
##  Box-Pierce test
##
## data:  residuals
## X-squared = 7.0325, df = 10, p-value = 0.7224
```

We do not reject the Null of No Autocorrelation. The AR(1) model captures most of the linear dependence existing in the GDP growth rate.

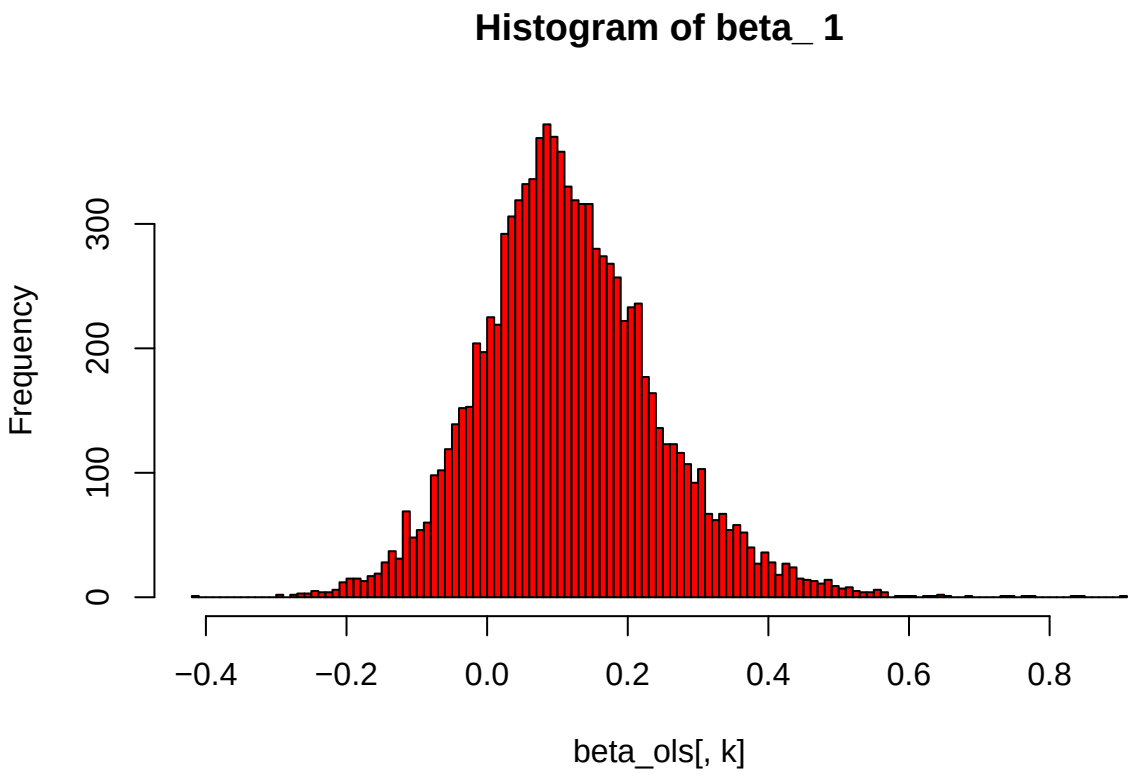
OLS estimator is biased

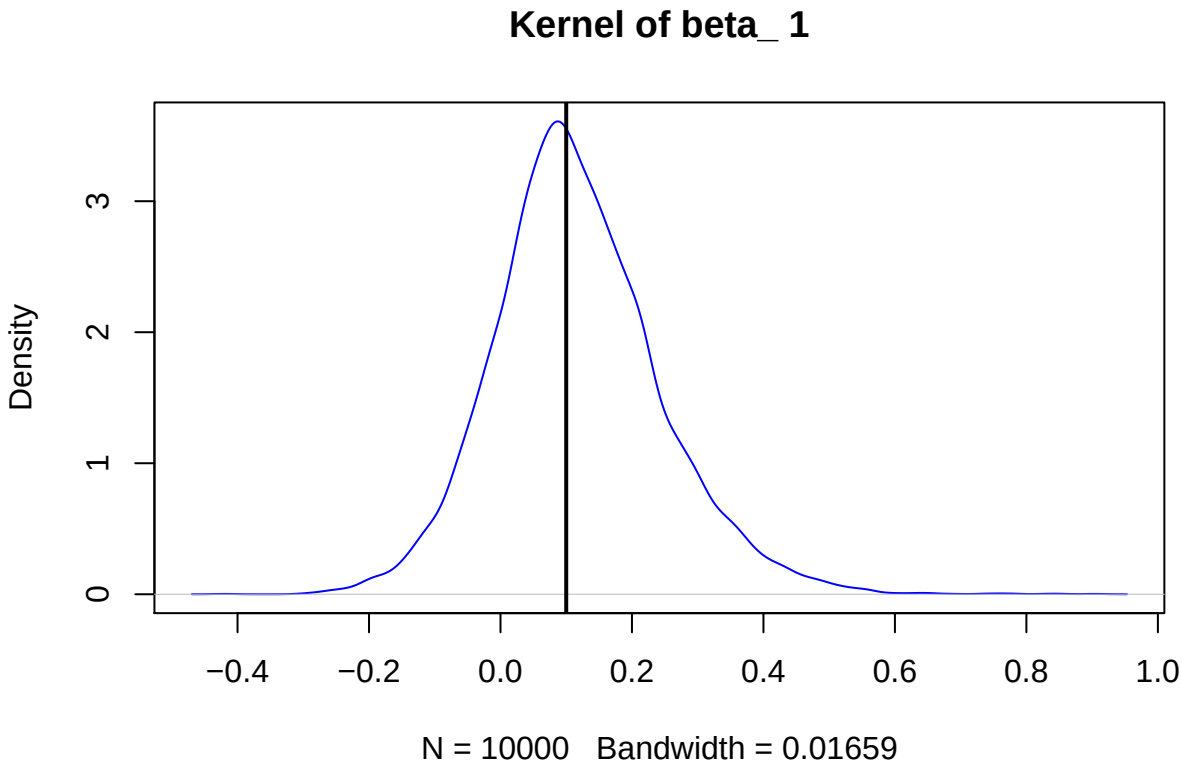
We turn to the OLS properties. We start by showing that the OLS estimator is biased. To highlight that property, we need to be in a controlled environment. Therefore, we will simulate the AR model to ensure that the error term are i.i.d.$(0,\sigma^2)$.

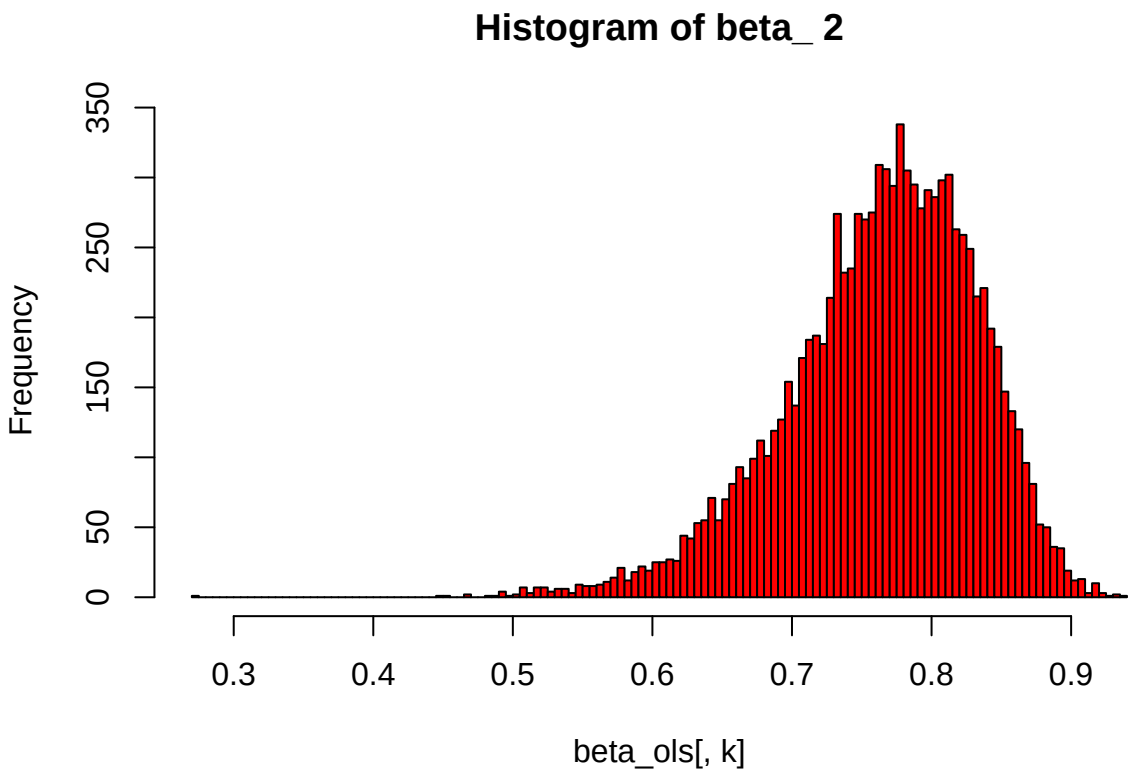
```
OLS_bias = function(T,beta_true,sigma_sq=1,nb_sim=1e4,dist=1){
  K = length(beta_true)
  if(K!=2){
    stop("Dimension of beta should be equal to two")
  }
  X = array(1,dim=c(T-1,2))
  beta_ols = matrix(0,nb_sim,2)
  for(i in 1:nb_sim){
    y = numeric(T)
    for(t in 2:T){
        if(dist==1){
          y[t] = beta_true[1] + y[t-1]*beta_true[2] + + rnorm(1)*sqrt(sigma_sq) ## Simulation with a No
        }else{
          y[t] = beta_true[1] + y[t-1]*beta_true[2] + + rt(1,10)*sqrt(sigma_sq) ## Simulation with a st
        }
    }
    X[,2] = y[1:(T-1)]
    y = y[2:T]
    beta_ols[i,] = solve(t(X)%*%X)%*%t(X)%*%y
  }
  for(k in 1:K){
    title = paste("Histogram of beta_",as.character(k))
    hist(beta_ols[,k],breaks=100,col="red",main=title)
    plot(density(beta_ols[,k]),col="blue",main=paste("Kernel of beta_",as.character(k)))
    abline(v=beta_true[k], lwd=2)
  }

  title = list()
  df_ols = matrix(0,nb_sim,K)
  for(k in 1:K){
    title[k] = paste("Bias of beta_",as.character(k),sep="")
    df_ols[,k] = beta_ols[,k]-beta_true[k]
  }
  df_ols = data.frame(df_ols)
  colnames(df_ols) = title
  print(summary(df_ols))

  return(df_ols)
}
```

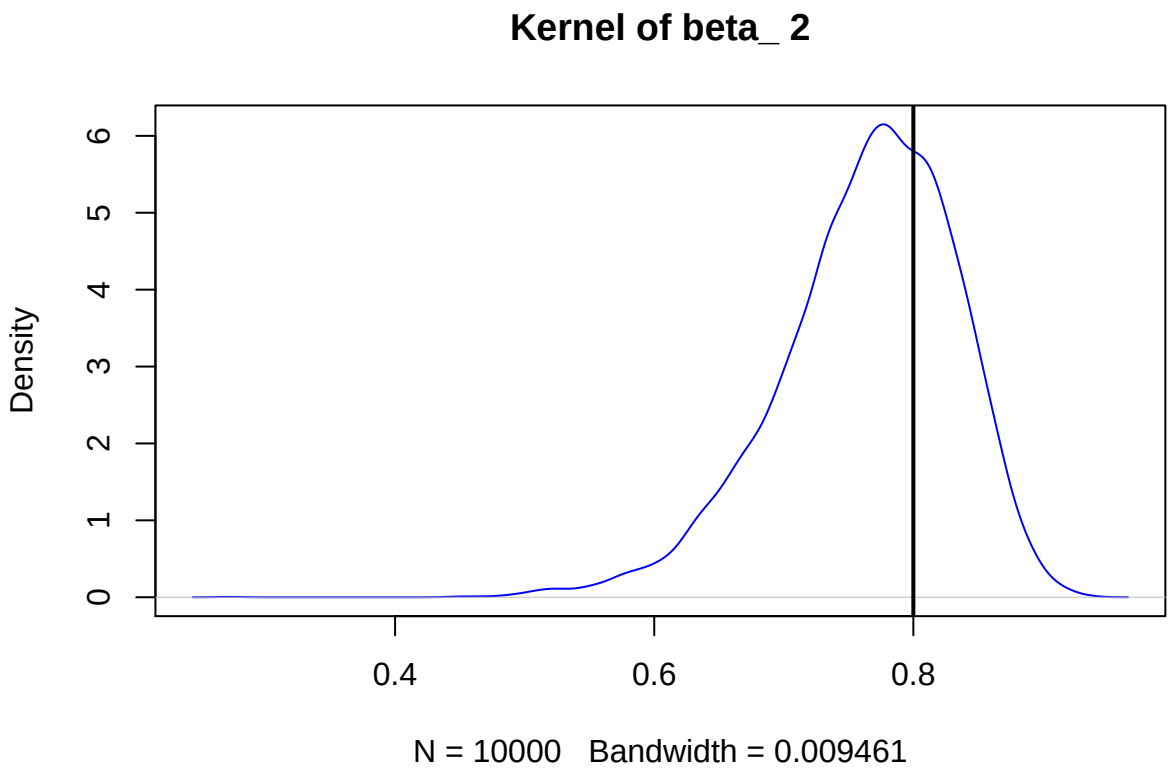
The *OLS_bias* function generates *nb_sim* dependent variables of size Tx1 from an AR(1) model with error terms distributed as a Normal distribution if *dist=1* or as a student distribution if *dist=2*. For each generated dependent variable, the OLS estimates are computed and compared to the true coefficients.

```
df_ols = OLS_bias(100,c(0.1,0.8))
```


Histogram of beta_ 1
Frequency
0
100
200
300
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
beta_ols[, k]


Kernel of beta_ 1
Density
0
1
2
3
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
N = 10000 Bandwidth = 0.01659


Histogram of beta_ 2
Frequency
0
50
150
250
350
0.3
0.4
0.5
0.6
0.7
0.8
0.9
beta_ols[, k]

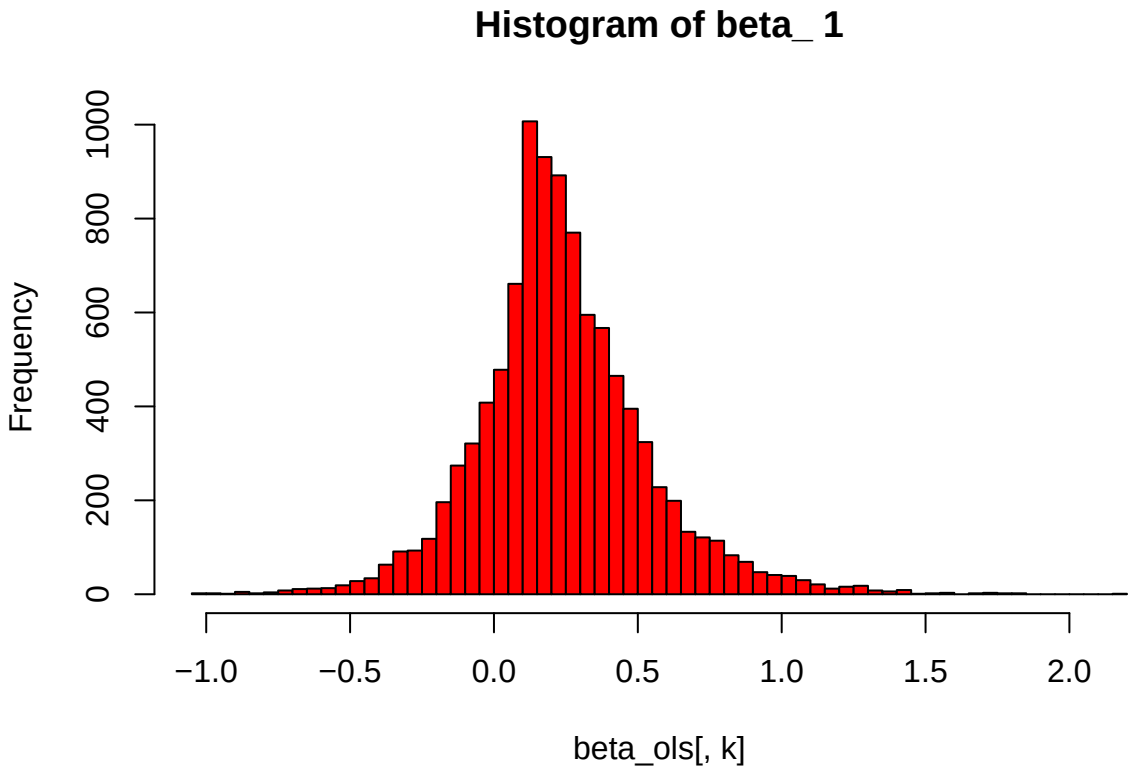


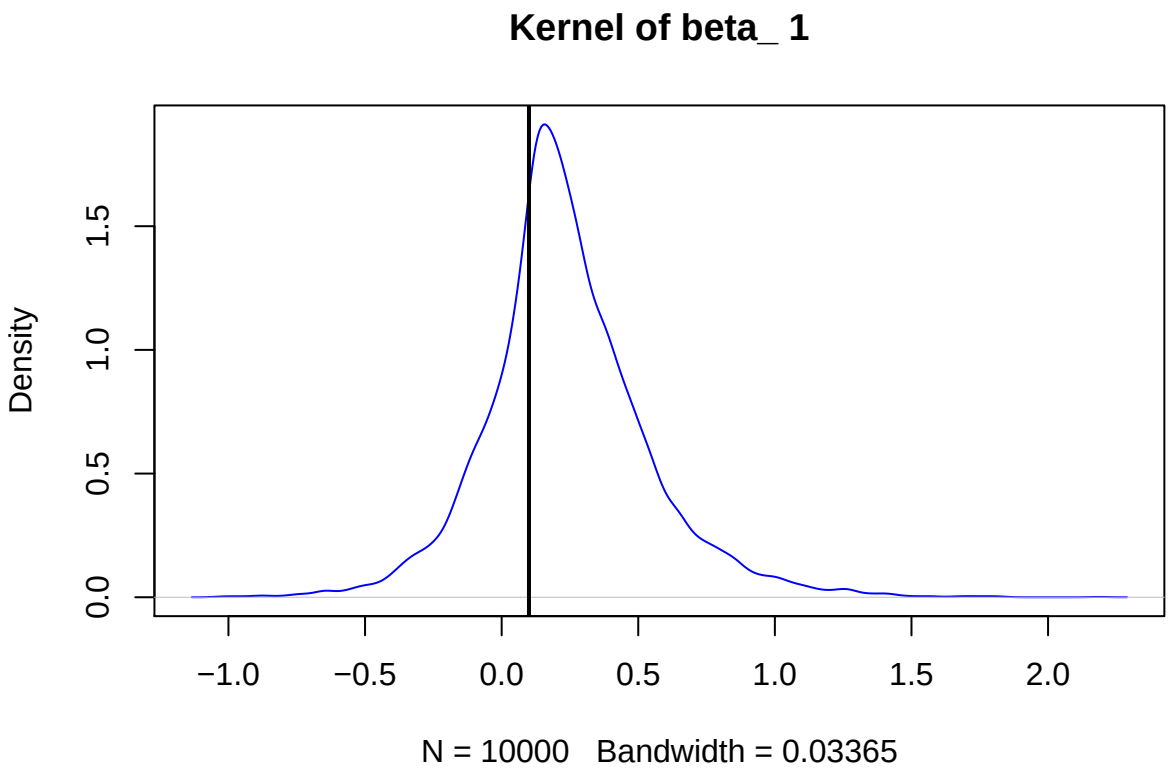

```
##  Bias of beta_1      Bias of beta_2
##  Min.   :-0.519742   Min.   :-0.52772
##  1st Qu.:-0.064929   1st Qu.:-0.07573
##  Median : 0.006664   Median :-0.02868
##  Mean   : 0.016704   Mean   :-0.03612
##  3rd Qu.: 0.090953   3rd Qu.: 0.01315
##  Max.   : 0.803195   Max.   : 0.13724
```

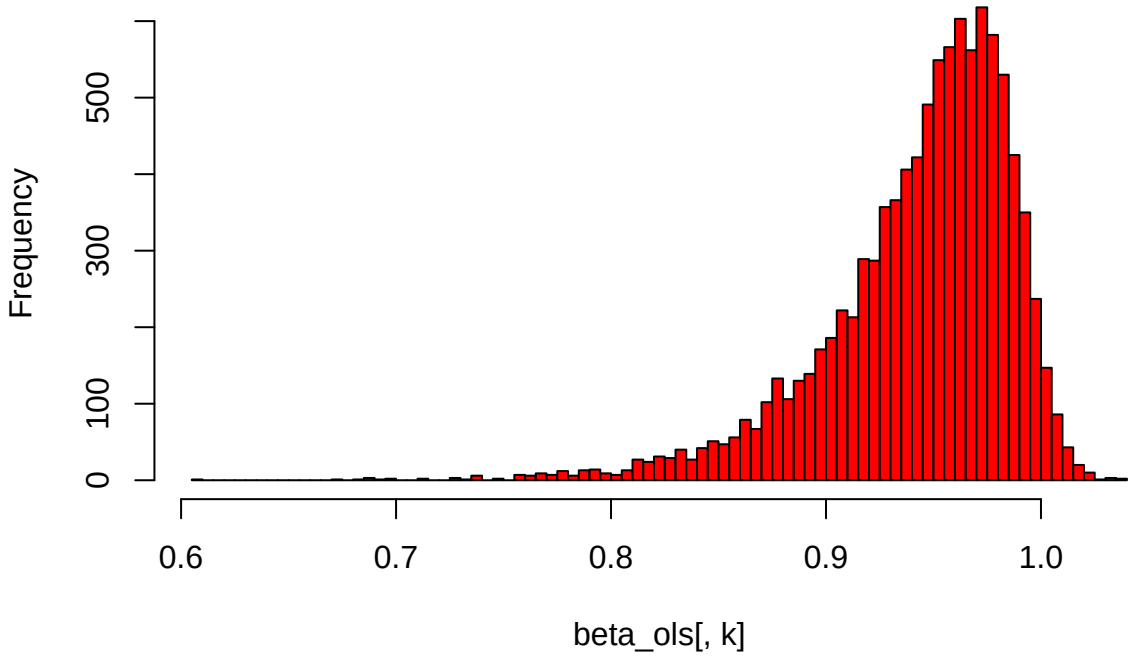
```
bias_beta = c(mean(df_ols$`Bias of beta_1`),mean(df_ols$`Bias of beta_2`))
```

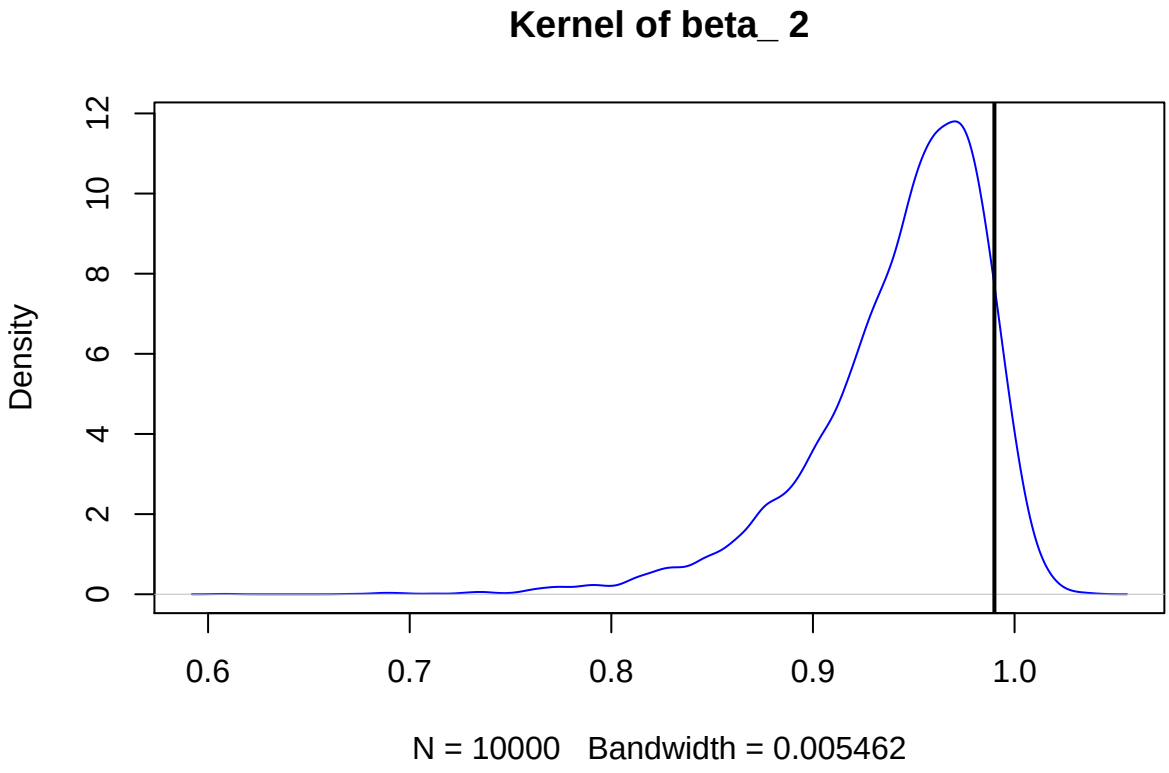
Even though the sample size is quite large (T=100), we observe that the OLS estimates are not centered around the true values. This bias problem is exacerbated when the persistence in the AR model increases.

```
df_ols = OLS_bias(100,c(0.1,0.99))
```


Histogram of beta_ 1
Frequency
0
200
400
600
800
1000
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
beta_ols[, k]


Kernel of beta_ 1
Density
0.0
0.5
1.0
1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
N = 10000 Bandwidth = 0.03365


Histogram of beta_ 2
Frequency
0
100
300
500
0.6
0.7
0.8
0.9
1.0
beta_ols[, k]




```
##  Bias of beta_1     Bias of beta_2
##  Min.   :-1.13301   Min.   :-0.38167
##  1st Qu.:-0.02611   1st Qu.:-0.06691
##  Median : 0.11032   Median :-0.03703
##  Mean   : 0.13818   Mean   :-0.04636
##  3rd Qu.: 0.29002   3rd Qu.:-0.01560
##  Max.   : 2.08775   Max.   : 0.04936
```

```
bias_beta = c(mean(df_ols$`Bias of beta_1`),mean(df_ols$`Bias of beta_2`))
```

OLS Consistency

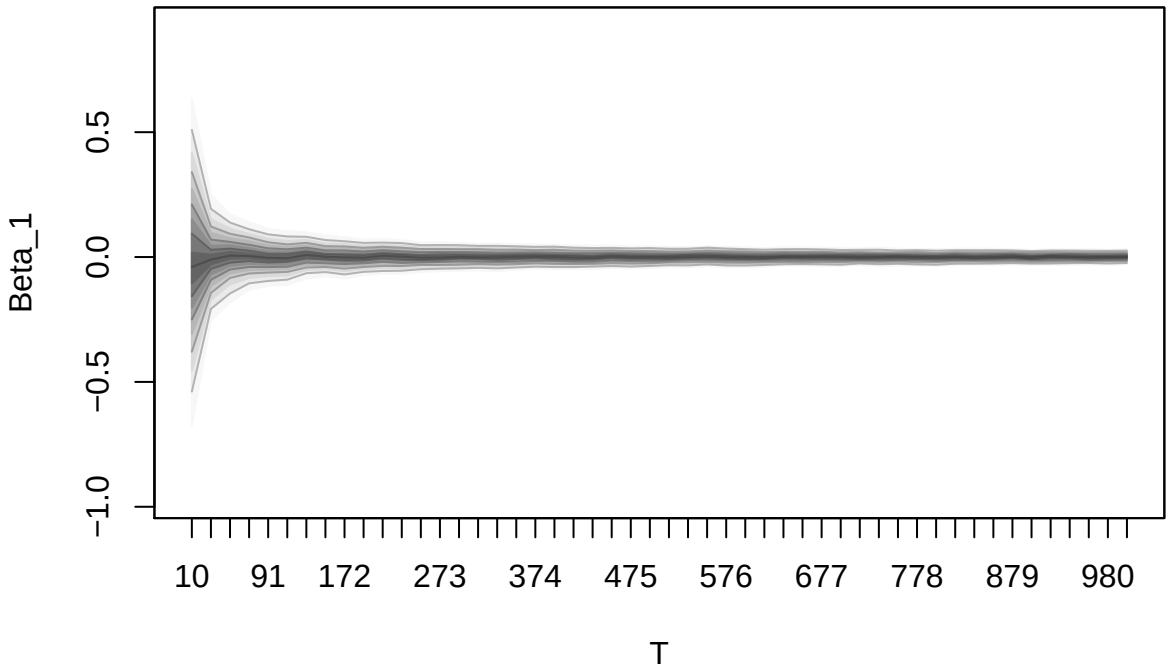
We now turn to the consistency property. Only the i.i.d assumption on the error terms and the fact that $|\beta_2| < 1$ are required to have the consistency property. We write a function to show that when the sample size increases, the OLS estimates shrink toward the true AR parameters.

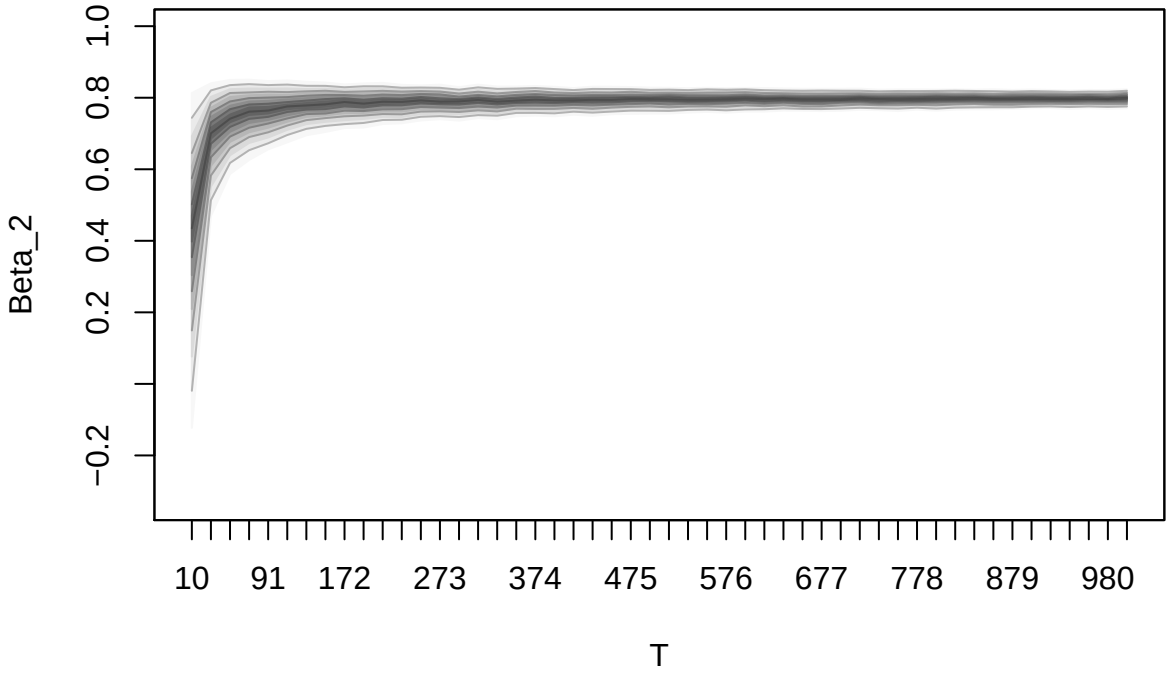
```
OLS_consistency = function(beta_true=c(0,0.8),sigma_sq = 2,nb_sim=1000){
  require(fanplot)
  K = length(beta_true)
  if(K!=2){
    stop("Dimension of beta should be equal to two")
  }
  T = 1000
  nb_iter = 50
  inter = ceiling(seq(from=10,to=T,length.out =nb_iter))
  df_student = 10
```

```
  seq_quant = seq(0.05, 0.95, 0.05)
  nb_quant = length(seq_quant)

  beta_ols = array(0,c(nb_sim,K,nb_iter))
  beta_quant = array(0,c(nb_quant,nb_iter,K))
  for(i in 1:nb_iter){
    T_cur = inter[i]
    y = numeric(T_cur)
    X = array(1,dim=c(T_cur-1,2))
    for(j in 1:nb_sim){
      for(t in 2:T_cur){
        y[t] = beta_true[1] + y[t-1]*beta_true[2] +  sqrt(sigma_sq)*rnorm(1)
      }
      X[,2] = y[1:(T_cur-1)]
      y_cur = y[2:T_cur]
      beta_ols[j,,i] = solve(t(X)%*%X)%*%t(X)%*%y_cur
    }
    # print(paste("Sample size of ",T_cur,sep=""))
    # mystats(beta_ols[,,i])
    beta_quant[,i,]= apply(t(beta_ols[,,i]), 1, quantile, probs = seq_quant,  na.rm = TRUE)
  }
  df_quant = list()
  for(k in 1:K){
    a = beta_quant[,,k]
    Name = paste("Beta_",k,sep="")
    plot(a[10,],ylab=Name,xlab="T",type="l",ylim=c(min(a[1,])-0.1,max(a[nb_quant,]+0.1)),xaxt = "n")
    pal <- colorRampPalette(c("gray20", "gray90"))
    fan(data=beta_quant[,,k],probs=seq_quant,sim.data=FALSE,alpha = 0.3,rlab=NULL,fan.col = pal,xaxt =
    axis(1, at=1:nb_iter, labels=inter)
    df_quant[[k]] = data.frame(beta_quant[,,k])
    colnames(df_quant[[k]]) = apply(as.matrix(inter),1,paste,"T",sep=" ")
  }

  return(df_quant)

}
```

```
quant_beta = OLS_consistency(beta_true = c(0,0.8),sigma_sq = 0.5) ## No Strict exogeneity .
```

```
## Loading required package: fanplot
```


Beta_1
0.5
0.0
−0.5
−1.0
10
91
172
273
374
475
576
677
778
879
980
T


Beta_2
1.0
0.8
0.6
0.4
0.2
−0.2
10
91
172
273
374
475
576
677
778
879
980
T