Assignment 1, due 22. March 2023

Topics in Advanced Time Series Econometrics

Prof. Dr. Marc Paolella

presented by

Migle Kasetaite: 21-733-779

Kilian Sennrich: 18-051-060

Abstract

This is our submission of the first assignment to the Topics in Advanced Time Series Econometrics course taught by Prof. Dr. Paolella (FS23). In the first part of the assignment, we compare the theoretical F-statistic and R^2-statistic with empirically observed ones via simulation. In a second part, we assess the behaviour of R^2 under an increasing number of random regressors. In the third part, we simulate an AR(1) process in which we concretely model the error term ϵ_t, and estimate the parameters using a fast iteration procedure based on Ordinary Least Squares & Generalized Least Squares.

1 Simulation Study - Theoretical F & R^2 Statistic Vs. Empirical Counterparts

In this exercise a simulation study was conducted to compare the behaviour of the R^2 and F statistics that were observed via simulations in comparison with the theoretical ones for $\beta = 0$ and $\beta = 1$ and also different σ's ($\sigma = 1, 2, 4$). R^2 is statistic that measures the fraction of the variability of Y taken into account by the linear regression model that includes a constant, compared to use of just a constant. It provides a measure of the extent to which the regressors "explain" the dependent variable over and above the contribution from just the constant term (Paolella, 2018).
The simulation experiment was conducted with 100'000 trials for each σ and β. We sticked to the same (50 x 4) X matrix for all the trials. $X = [1|R]$ was generated where $1 = [1_1, \ldots, 1_{50}]^T$, and $R = [r_{1:50,3}]$ with $r_{1:50,3} \overset{iid}{\sim} \mathcal{N}(0,1)$. That is, 1 denotes a vector of ones and R is a matrix of random variates from a Gaussian normal distribution with location 0 and scale 1. We calculated the true Y as $Y = X\beta + \epsilon$, with $\beta = [0,0,0,0]^T$ and later $\beta = [1,1,1,1]^T$ and $\epsilon = [\epsilon_1, \ldots, \epsilon_{50}]^T$ with $\epsilon \overset{iid}{\sim} \mathcal{N}(0,\sigma)$, where $\sigma = 1, 2, 4$. $\hat{Y}$ was computed using the following formula: $\hat{Y} = X\hat{\beta}$, with $\hat{\beta} = (X^TX)^{-1}X^T\epsilon$. $\hat{\beta}$ follows from the fact that $\hat{\beta} = \beta + (X^TX)^{-1}X^T\epsilon$ (Paolella, 2018). The empirical R^2-statistic was calculated according to $R^2 = \frac{ESS}{TSS}$ where $ESS = \sum_{t=1}^{100}(\hat{Y}_t - \bar{Y})^2 = \dot{Y}^T\dot{Y}$, where $\dot{Y} = \hat{Y} - \bar{Y}$ and $TSS = \sum_{t=1}^{100}(Y_t - \hat{Y})^2 + ESS = \ddot{Y}^T\ddot{Y} + ESS$, where $\ddot{Y} = Y - \hat{Y}$. Theoretical R^2 follows Beta distribution, $R^2 \sim Beta(\frac{k-1}{2}, \frac{T-k}{2})$. Similarly, empirical F-statistic is computed using formula: $F = \frac{T-k}{k-1}\frac{R^2}{1-R^2}$, and its theoretical counterpart follows F distribution, $F \sim F(J, T-k)$ (Paolella, 2018).
From the Figure 1 we can see that when β is 0, R^2 and F statistics are the same for different σ's, R^2 is around 0.1 and F is around 0.7. We also observe that theoretical and empirically obtained statistics are almost identical. For $\beta = 1$ we only computed results via simulation. From the Figure 2 we can see that R^2 explains more variation in the data and here σ also plays a role, the bigger σ, the smaller R^2 gets. For F statistic one can see that with the increase in values of σ, F values are increasing, the mean moves towards 0 and plot under the curve gets more skewed.
The Python-Code for this experiment can be found in Code-Section 1.

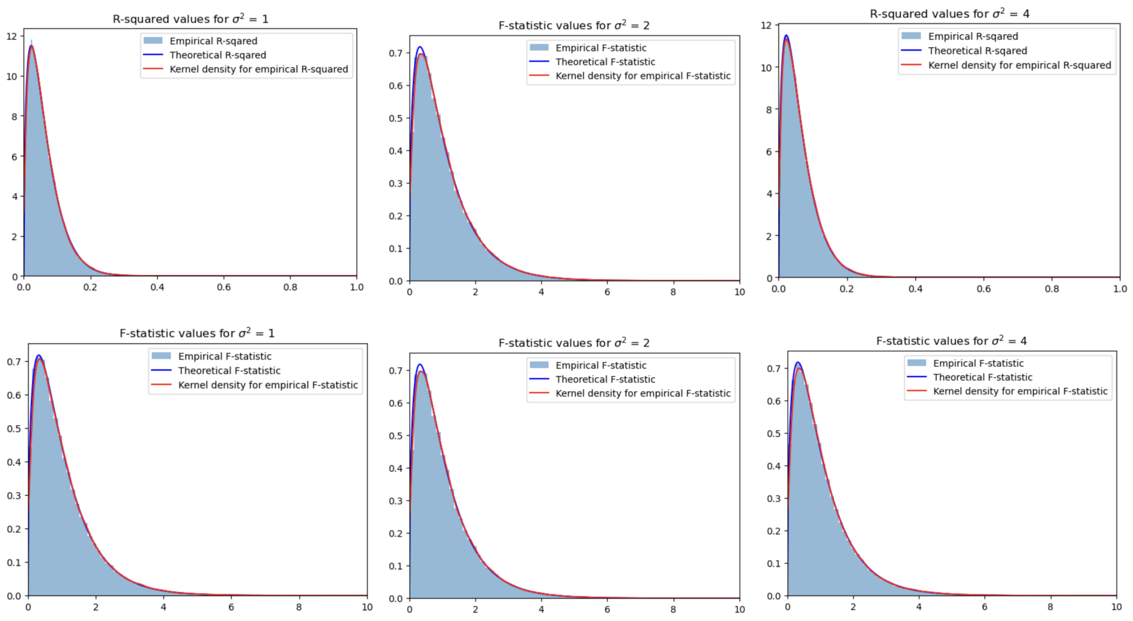

Figure 1: Density plots for theoretical F and R-squared statistics compared with the empirically obtained ones. Here $\beta = 0$ and $\sigma = 1, 2$ and 4.

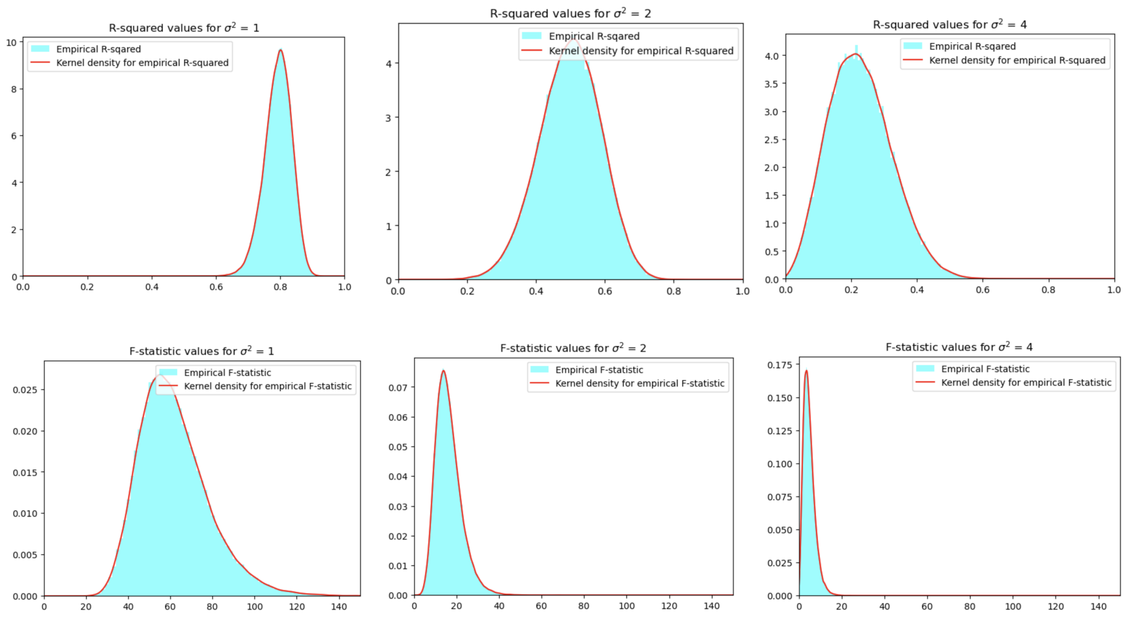

Figure 2: Density plots for empirically obtained F and R-squared statistics. Here $\beta = 1$ and $\sigma = 1, 2$ and 4.

2 Simulation Study - Behaviour of the R^2-Statistic Under Increasing Random Regressors

A simulation study was conducted to assess the behaviour of the R^2 statistic under an increasing number of random regressors. R^2 is the amount of variance of the regressand explained by the regressors. Since all variables were randomly generated, the R^2 should in theory, for any number of regressors, be 0. However, since the empirically obtained correlations between the regressors and the regressand range from between -0.05 to 0.05 (approximately), the observed R will be larger than 0.
It should also be noted that the R^2 statistic does not correct for a greater number of parameters (beta), as is the case with the AIC (Sakamoto et al., 1986) or BIC (Schwarz, 1978). Accordingly, it is expected that with an increasing number of regressors, the R^2, even if it should theoretically stay 0, will slowly increase. This fact must be taken into account when comparing models with different numbers of regressors. It is therefore recommended to use AIC or BIC for such comparisons. These theoretical results were confirmed by our simulation study. Figure 3 shows that the R^2 statistic increases with an increasing number of regressors. Further, it can also be seen that the standard deviation of the estimated R^2 statistics increases with increasing number of regressors. We see the reason for this in fact is that the randomness in the creation of the regressors in the X-matrix "sums up".
The simulation experiment was conducted with 10'000 trials for every k, where $\{k | k \in \mathbb{N}, k \geq 2, k \leq 10\}$. During each trial, a matrix $X = [1|R]$ was generated where $1 = [1_1, \ldots, 1_{100}]^T$, and $R = [r_{1:100,1}, \ldots, r_{1:100,k-1}]$ with $r_{1:100,i} \overset{iid}{\sim} \mathcal{N}(0,1)$. That is, 1 denotes a vector of ones and R is a matrix of random variates from a Gaussian normal distribution with location 0 and scale 1. True Y was calculated according to $Y = X\beta + \epsilon$, with $\beta = [0_1, \ldots, 0_{k+1}]^T$ and $\epsilon = [\epsilon_1, \ldots, \epsilon_{100}]^T$ with $\epsilon \overset{iid}{\sim} \mathcal{N}(0,2)$. The estimation of Y, $\hat{Y}$ was computed according to $\hat{Y} = X\hat{\beta}$, with $\hat{\beta} = (X^TX)^{-1}X^T\epsilon$. $\hat{\beta}$ follows from the fact that $\hat{\beta} = \beta + (X^TX)^{-1}X^T\epsilon$ (Paolella, 2018). Notice that the same ϵ was used as for Y and that β is a vector of zeros, i.e. the addition can be omitted. The R^2-statistic was finally calculated according to $R^2 = \frac{ESS}{TSS}$ where $ESS = \sum_{t=1}^{100}(\hat{Y}_t - \bar{Y})^2 = \dot{Y}^T\dot{Y}$, where $\dot{Y} = \hat{Y} - \bar{Y}$ and $TSS = \sum_{t=1}^{100}(Y_t - \hat{Y})^2 + ESS = \ddot{Y}^T\ddot{Y} + ESS$, where $\ddot{Y} = Y - \hat{Y}$ (Paolella, 2018).
The Python-Code for this experiment can be found in Code-Section 2.

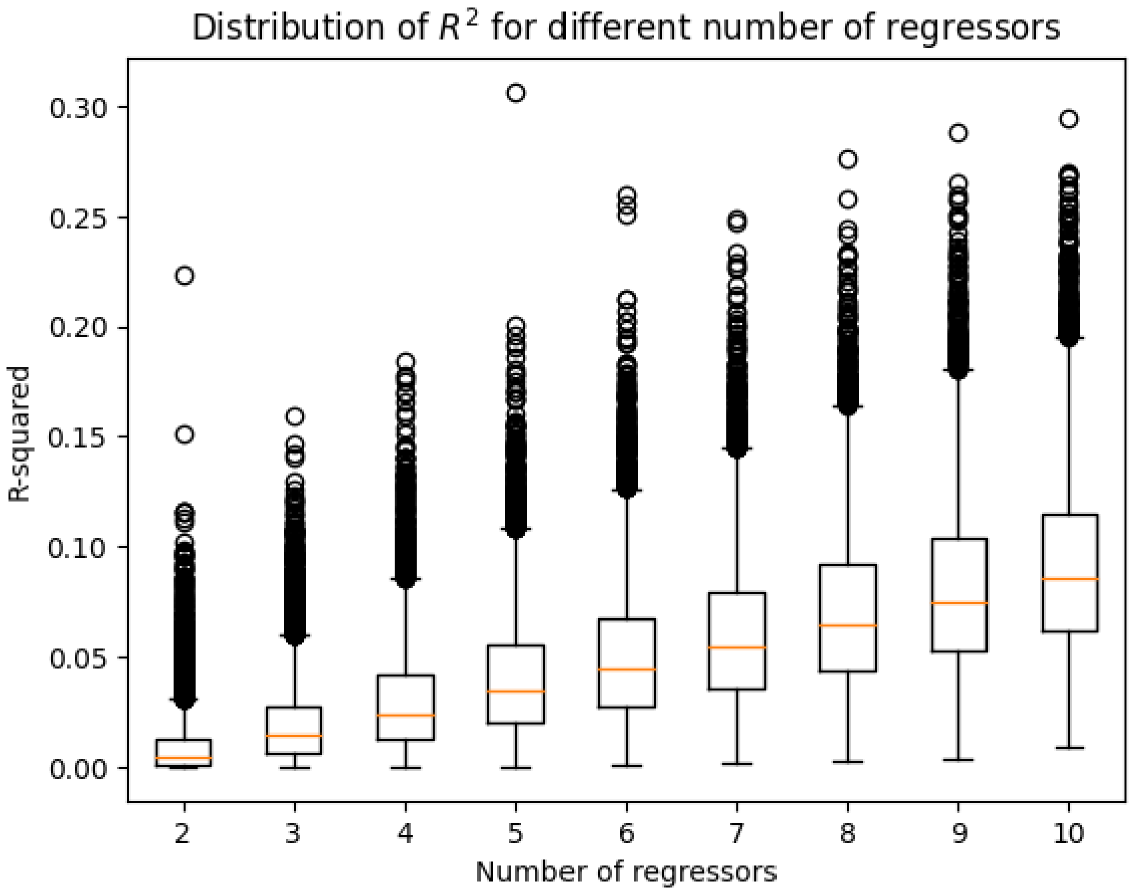

Figure 3: Boxplots for the distributions of R^2 for an increasing number of random regressors. Even though all regressors are independent of the regressand, the R^2 increases for an increasing number of regressors.

3 Simulation Study - Modelling and Estimation of the Error Term ϵ in the AR(1)-Process.

3.1 Historical and contextual clarifications

In this last part, a simulation study was conducted to assess the estimation of the free parameters of an AR(1) process. In this respect, we study a regression model, where the error term is concretely modeled. The autoregressive part of the model, $Y = X\beta + \epsilon_t$ is thus expanded using $\epsilon_t = a\epsilon_{t-1} + U_t$, with $U_t \overset{iid}{\sim} \mathcal{N}(0, \sigma^2)$. So, trivially put together, we use the following model.

$$Y = X\beta + a\epsilon_{t-1} + U_t, U_t \overset{iid}{\sim} \mathcal{N}(0, \sigma^2) \tag{1}$$

The model has k + 2 free parameters, where k represents the number of columns in the X matrix. To be estimated are consequently $[\beta_0, \dots, \beta_k]^T, a,$ and σ^2. To estimate these parameters, we use an iterative procedure that has its seeds within the *Cochrane-Orcutt* estimation procedure (Cochrane & Orcutt, 1949). This procedure was developed to correct the regression model for serial autocorrelation in the error terms and was originally applied in test theory. Under the assumptions of stationarity and that the available data are based on a first-order autoregressive process, the original model can be quasi-differentiated as follows:

$$Y_t - aY_{t-1} = \beta_0(1-a) + (X_t - aX_{t-1})\beta_{1:k} + \epsilon_t \tag{2}$$

For the estimation of β parameters, the sum of squared residuals ϵ_t^2 can be minimized, constrained by a.
By the quasi-difference in (2), the first observation is lost (since no previous t-1 is known for the first observation at time t), which can become problematic in small samples. To counteract this problem, Prais and Winsten (1954) developed an extended version of Cochrane-Orcutt's procedure. More on this follows further below. First, however, we would like to discuss the estimation of a in the Cochrane-Orcutt procedure. The coefficient for autocorrelation a can be estimated by regressing the untransformed model and obtaining the residuals $\hat{\epsilon}_t$, then regressing $\hat{\epsilon}_t$ on $\hat{\epsilon}_{t-1}$, i.e. $\hat{\epsilon}_t = a\hat{\epsilon}_{t-1} + \dot{e}_{t-1}$, where $\dot{e}_{t-1}$ is the newly estimated residual of the residual-regression. $\dot{e}_{t-1}$ can then be autoregressed again until no significant change in the value of a is detected.
Notice that the Cochrane-Orcutt estimation procedure looses the first observation. To counteract this phenomenon, Prais and Winsten (1954) presented an extended version of the procedure. In addition to the Cochrane-Orcutt, they make the following transformation at t = 1: $\sqrt{1-a^2}Y_1 = \beta_0\sqrt{1-a^2} + (\sqrt{1-a^2}X_1)\beta_{1:k} + \sqrt{1-a^2}\epsilon_1$. Notice the origin of this form in (2). Do derive the estimate for a, the recursive procedure of Cochrane-Orcutt (above) can be used. To estimate the regression weights $\hat{\beta}$, the Generalized Least Squares (GLS) procedure is used. First note that the autocovariance function of the error term is $cov(\epsilon_t, \epsilon_{t+h}) = a^h var(\epsilon_t)$. Considering a white noise process with variance 1, then $var(\epsilon_t) = \frac{1}{1-a^2}$. It is straight forward to see that

$$\Sigma = \begin{bmatrix} \frac{1}{1-a^2} & \frac{a}{1-a^2} & \frac{a^2}{1-a^2} & \cdots & \frac{a^{T-1}}{1-a^2} \\ \frac{a}{1-a^2} & \frac{1}{1-a^2} & \frac{a}{1-a^2} & \cdots & \frac{a^{T-2}}{1-a^2} \\ \frac{a^2}{1-a^2} & \frac{a}{1-a^2} & \frac{1}{1-a^2} & \cdots & \frac{a^{T-3}}{1-a^2} \\ \cdots & \cdots & \cdots & \ddots & \cdots \\ \frac{a^{T-1}}{1-a^2} & \frac{a^{T-2}}{1-a^2} & \frac{a^{T-3}}{1-a^2} & \cdots & \frac{1}{1-a^2} \end{bmatrix} \tag{3}$$

Now having the covariance matrix, $\hat{\beta}$ is straight forward to compute using GLS:

$$\hat{\beta} = (X^T\Sigma^{-1}X)^{-1}(X^T\Sigma^{-1}Y) \tag{4}$$

We have now seen where the roots of the parameter estimates for AR(1) processes with serial autocorrelation lie. On this foundation, we will now justify the method described by Paolella (2018) on page 225. To this end, we elaborate on Feasible Generalized Least Squares (FGLS). FGLS is a method to estimate $\hat{\beta}_{GLS}$ when the covariance matrix Σ of the error terms is unknown. The estimation in FGLS proceeds in two stages.
In a *first* stage, the model is estimated using a consistent estimator, such as OLS. Notice that the error terms are correlated here, i.e. OLS is inefficient (Mathematical inefficiency leads to the free parameters containing bias, which makes them useless in inferential statistics and potentially leads to inaccurate forecasts in prediction). The residuals of the OLS estimation are then used to construct a covariance matrix of the error terms (Johnston, 1972). Notice that in (1), the residuals are concretely modeled. From the elaboration on the Prais and Winsten (1954) procedure above, it is evident that in order to generate the covariance matrix in (3), the parameter a must first be estimated. a can be estimated from the residuals as follows:

$$\hat{a}_{LS} = \frac{\sum_{t=1}^{T} \hat{\epsilon}_t \hat{\epsilon}_{t-1}}{\sum_{t=0}^{T-1} \hat{\epsilon}_t^2} = \frac{\hat{E}_{T-1}^T \hat{E}_T}{\hat{E}_{T-1}^T \hat{E}_{T-1}} = \frac{\hat{E}^T D_{T-1}^T D_N \hat{E}}{\hat{E}^T D_{T-1}^T D_{T-1} \hat{E}} \tag{5}$$

with $\hat{E} = [\hat{\epsilon}_1, \dots, \hat{\epsilon}_T]^T$, $D_t = [0|I_T]$, $D_{t-1} = [I_T|0]$ and $0 = [0_1, \dots, 0_T]^T$ (Paolella, 2018). From this step, the variance σ^2 of U_t in (1) can be calculated straight forward as (Paolella, 2018):

$$\hat{\sigma}_{\text{LS}}^2 = \frac{\sum_{t=1}^{T} \left(\hat{E}_t - \hat{a}_{\text{LS}} \hat{E}_{t-1} \right)^2}{T-1}. \tag{6}$$

Now having the covariance matrix constructed, the *second* stage in FGLS is to estimate autocorrelation robust estimates for the free parameters. This is done by iterating between (5) and (4), each time recalculating the residuals, $\hat{a}$ and $\hat{\beta}$. This second stage can be exited as soon as sufficient convergence is achieved.

3.2 Simulation Study

To assess the estimation of the parameters in the AR(1) process with autocorrelated error terms, we conducted a simulation study. We randomly generated $X = [1|R]$ with $1 = [1_1, \dots, 1_{100}]^T$, and $R = [r_{1:T,1}, \dots, r_{1:T,k-1}]$ with $r_{1:T,i} \overset{iid}{\sim} \mathcal{N}(0,1)$ and $T \in \{20, 50\}$. In P repetitions, where $P \in \{10'000, 1000\}$ we simulated an AR(1) process according to (1). The following specification was used to generate the process: $\beta = [0,0,0,0,0]^T, a \in \{0.0, 0.2, 0.4, 0.6, 0.8, 0.99\}$ and $U \sim \mathcal{N}(0,1)$. The first error term ϵ_0, on $(-1,1)$, is assumed to be a realisation of the unconditional distribution $U_t \overset{iid}{\sim} \mathcal{N}(0, \sigma^2/1 - a^2)$ (Paolella, 2018). Following the generation of the AR(1) process, $\hat{\beta}_{LS}$ was computed and $\hat{a}_{LS}$ and $\hat{\sigma}_{LS}^2$ were derived according to (5) and (6) from the resulting residuals. $\hat{a}_{LS}$ was then used to compute Σ from (3). Next, GLS in (4) was used to derive $\hat{\beta}_{GLS}$. From the residuals of the GLS, again $\hat{a}$, $\hat{\Sigma}$ and GLS were computed. This was done until convergence or non-convergence. Convergence was defined by the parameters not changing from one iteration to the next when rounded to 5 decimal places. Non-convergence was defined as a maximum number of iterations. Depending on a, this maximum number of iterations was adjusted between $\{100, 500, 1000\}$, such that the run times remained feasible. The implementation of the covariance matrix was done using dynamic programming and can be found in Code Section 3. The implementation of the experiment can be found in Code Section 4.

3.3 Results

The simulation study was divided into two parts and results are presented here. First, we examined how the estimation process behaves with small samples. For this purpose, we compared the two sample sizes T = 20 and T = 50. For both sample sizes, a of the simulated AR(1) process was set to 0 and σ^2 was set to 1. It was noticeable that at T = 20 about 13% of the 10'000 the runs did not converge after 500 iterations. In contrast, at T = 50, only 7 out of 10'000 did not converge. Figures 4 and 5 clearly show the different standard deviations around the true parameters. The estimates for the smaller sample are accompanied by larger standard deviations of the parameter estimates.

The second part of the analysis of the results is dedicated to investigating the effect of the parameter a in the generation of the AR(1) process on the parameter estimates. a can be interpreted as the "strength" of the autocorrelation in the error terms of the process.

The true AR(1) process was simulated with $a \in \{0.2, 0.4, 0.6, 0.8, 0.99\}$. All simulations were performed with 1000 repetitions. As can be seen in Figures 4 ($a = 0.2$) & 5 ($a = 0.4$), the estimates in AR(1) processes with moderate serial autocorrelation are very satisfactory. This is indicated by the relatively small standard deviations around the true parameter, by the mean estimates lying very close around the true parameters, and by the relatively normal distribution of the estimated parameters. Comparing estimates for $a = 0.4$ and $a = 0.6$ (Figure 6), one starts to notice a significant increase in the standard deviations of the estimates. Having a look at Figure 6, it is evident that parameter $\hat{a}$ is systematicatically underestimated. Nevertheless, the parameter estimates follow more or less a normal distribution. At $a = 0.8$, we found an interesting phenomenon. The estimation procedure seemed to almost never converge at a maximum of 500 iterations. In 1000 simulations, only 138 converged below 500 iterations. Looking at Figure 9, it becomes clear that the standard deviations of the estimates are still rather small. We are convinced that there is selection bias present: Only AR(1) processes where the error term U by chance has very good properties for convergence have managed to create a consistent estimate at all under 500 iterations. This has to be considered when interpreting Figure 9. So it seems that the estimation procedure for a = 0.8 converges only very slowly to the final estimate. We did not experience this phenomenon for $a = 0.99$. However, Figure 10 shows that the estimation procedure for $a = 0.99$ does not work properly. The estimates for $\hat{\beta}_0$ and $\hat{\sigma}^2$ appear meaningless; parameter $\hat{a}$ was clearly overestimated. The distribution of the estimates no longer has the form of a normal distribution.
In conclusion, the estimation procedure described above can only be used for AR(1) processes with low to moderate serial autocorrelation. In the case of strong serial autocorrelation, the estimates are strongly off the mark.

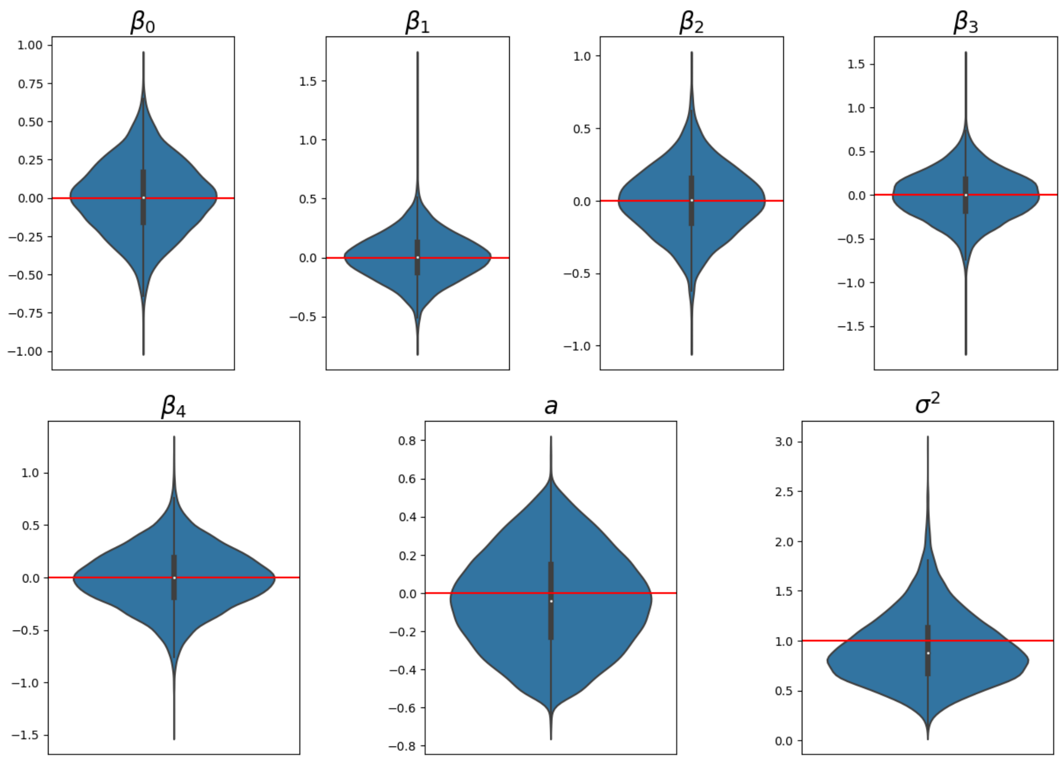

Figure 4: Violin plots of the distribution of the estimated parameters for simulation 3 for T = 20, a=0 and σ^2=1. The true parameter underlying the generation of the AR(1) process is indicated with the red line. The mean estimated parameter is indicated by the white dot. The standard deviations of the parameter estimates increase with smaller sample size.

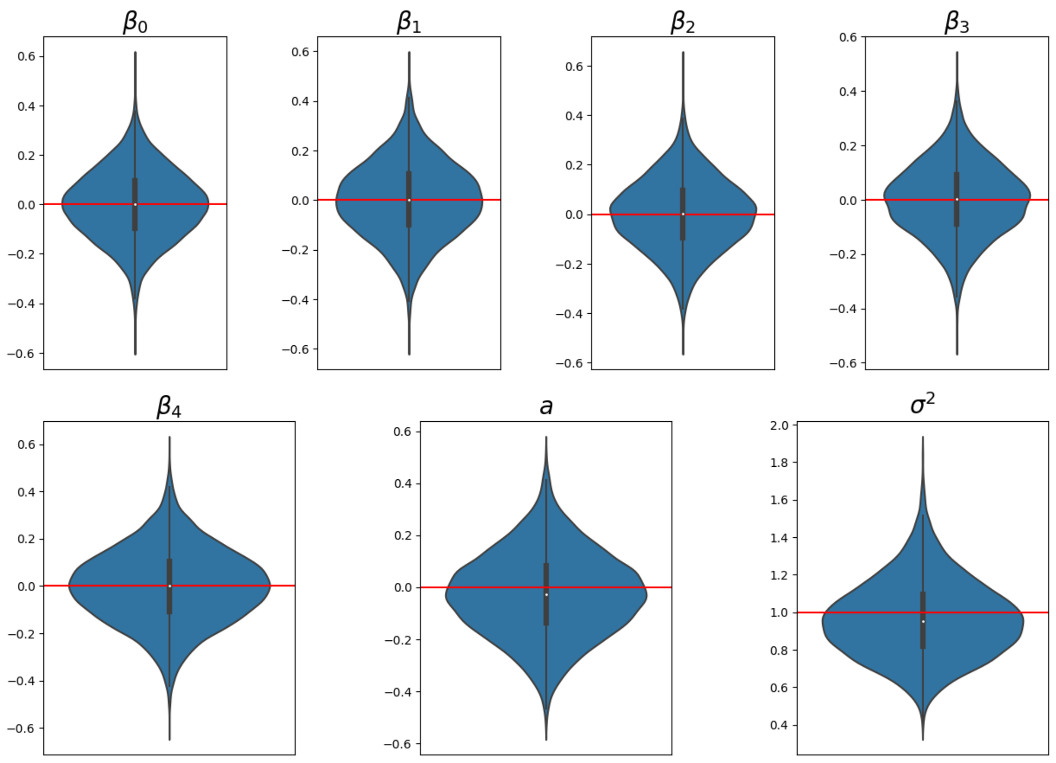

Figure 5: Violin plots of the distribution of the estimated parameters for simulation 3 for T = 50 a=0 and σ^2=1. The true parameter underlying the generation of the AR(1) process is indicated with the red line. The mean estimated parameter is indicated by the white dot. The standard deviations of the parameter estimates increase with smaller sample size.

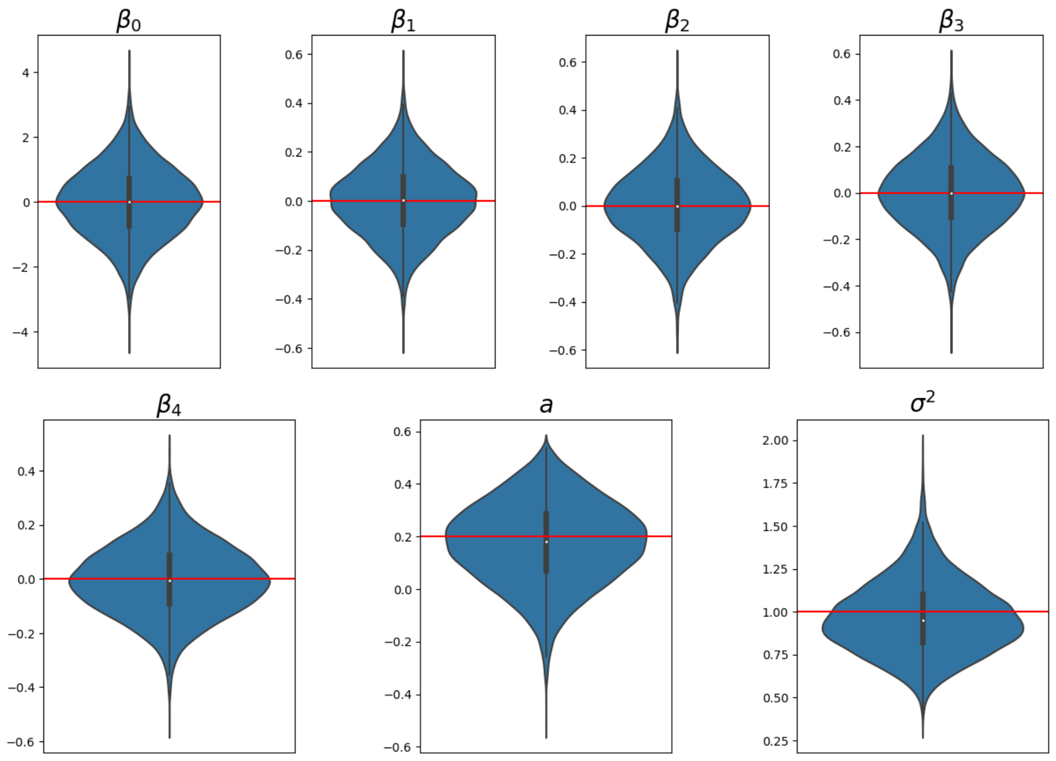

Figure 6: Violin plots of the distribution of the estimated parameters for simulation 3 for T = 50 and σ^2=1. The true a underlying the generating AR(1) process was 0.2. The true parameter underlying the generation of the AR(1) process is indicated with the red line. The mean estimated parameter is indicated by the white dot. The standard deviations of the parameter estimates increase with smaller sample size.

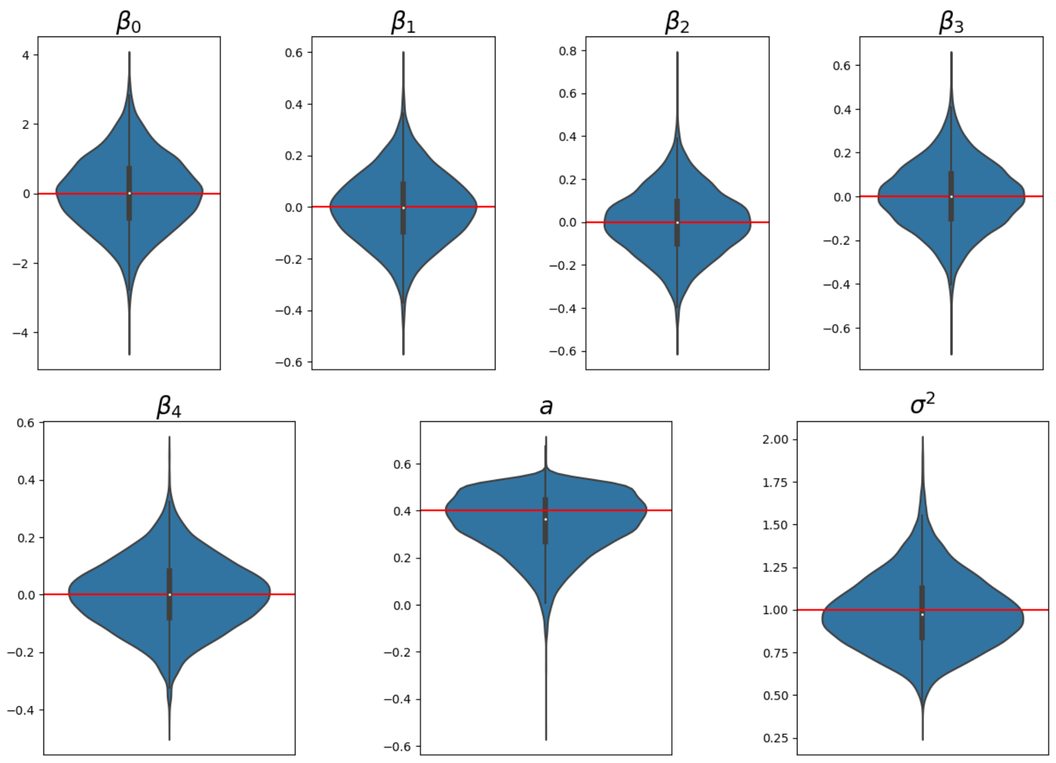

Figure 7: Violin plots of the distribution of the estimated parameters for simulation 3 for T = 50 and σ^2=1. The true a underlying the generating AR(1) process was 0.4. The true parameter underlying the generation of the AR(1) process is indicated with the red line. The mean estimated parameter is indicated by the white dot. The standard deviations of the parameter estimates increase with smaller sample size.

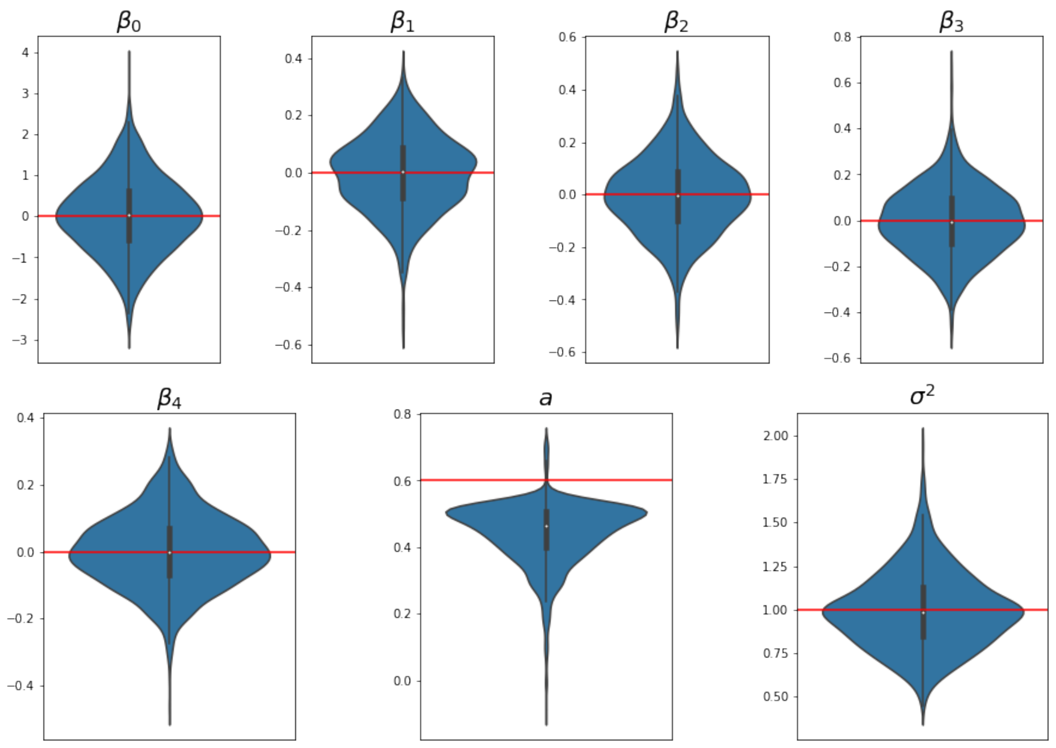

Figure 8: Violin plots of the distribution of the estimated parameters for simulation 3 for T = 50 and σ^2=1. The true a underlying the generating AR(1) process was 0.6. The parameter estimates are still acceptable, the estimate for a is off. The standard deviations of the estimates are rather large. The true parameter underlying the generation of the AR(1) process is indicated with the red line. The mean estimated parameter is indicated by the white dot. The standard deviations of the parameter estimates increase with smaller sample size.

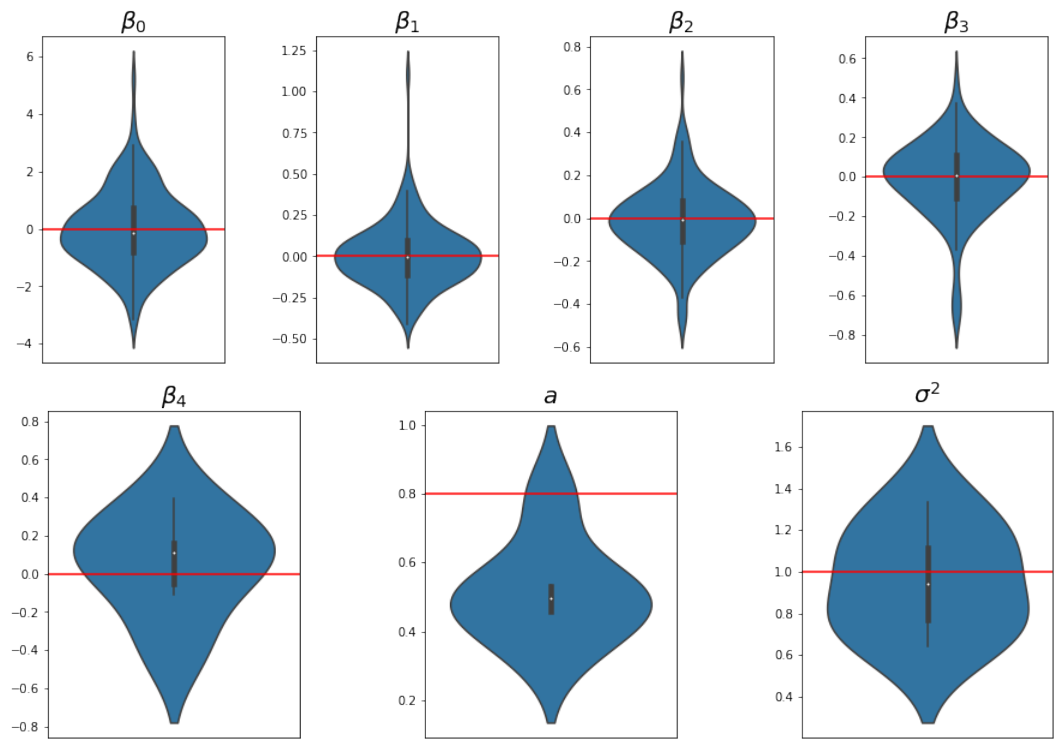

Figure 9: Violin plots of the distribution of the estimated parameters for simulation 3 for T = 50 and σ^2=1. The true a underlying the generating AR(1) process was 0.8. The true parameter underlying the generation of the AR(1) process is indicated with the red line. The mean estimated parameter is indicated by the white dot. The standard deviations of the parameter estimates increase with smaller sample size.

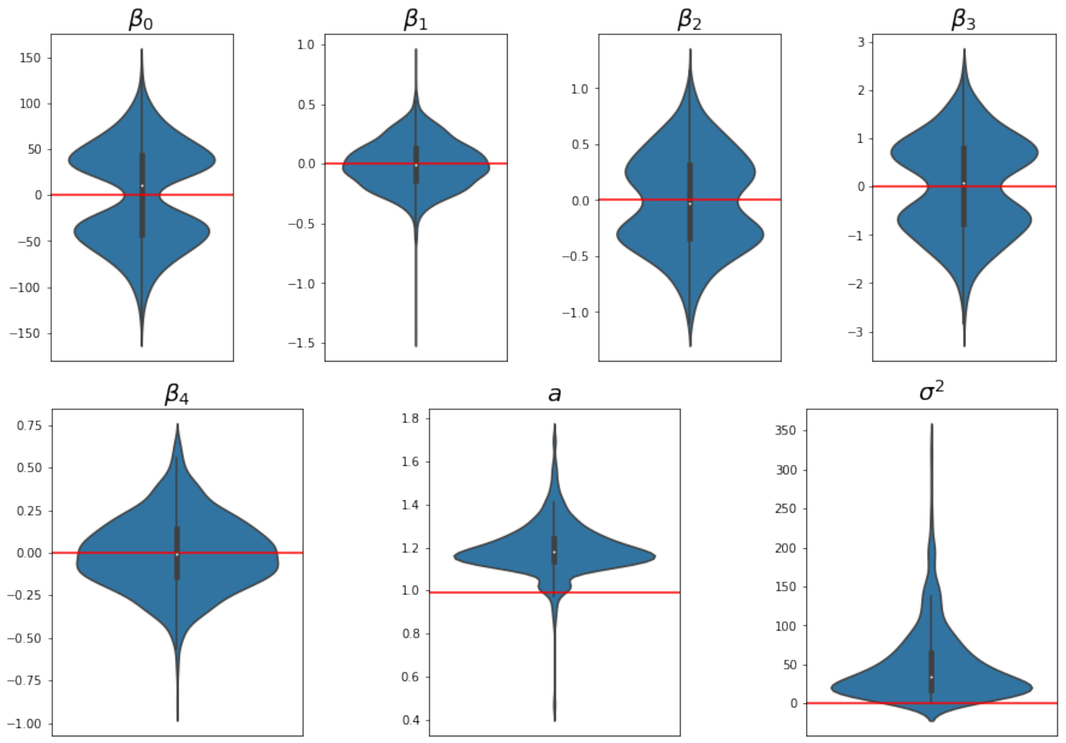

Figure 10: Violin plots of the distribution of the estimated parameters for simulation 3 for T = 50 and σ^2=1. The true a underlying the generating AR(1) process was 0.99. The parameter estimates are rather weak, true parameters and mean estimates are often times noticably different. The true parameter underlying the generation of the AR(1) process is indicated with the red line. The mean estimated parameter is indicated by the white dot. The standard deviations of the parameter estimates increase with smaller sample size.

4 Bibliography

Cochrane, D., & Orcutt, G. H. (1949). Application of least squares regression to relationships containing auto-correlated error terms. *Journal of the American statistical association*, 44(245), 32-61.

Johnston, John (1972). Econometric Methods (2nd ed.). *New York: McGraw-Hill.* pp. 259–265.

Paolella, M. S. (2018). *Linear models and time-series analysis: regression, ANOVA, ARMA and GARCH.* John Wiley & Sons.

Prais, S. J., & Winsten, C. B. (1954). *Trend estimators and serial correlation* (Vol. 383, pp. 1-26). Chicago: Cowles Commission discussion paper.

Sakamoto, Y., Ishiguro, M., & Kitagawa, G. (1986). *Akaike information criterion statistics.* Dordrecht, The Netherlands: D. Reidel, 81(10.5555), 26853.

Schwarz, G. (1978). The bayesian information criterion. Ann. Statist, 6, 461-464.

5 Appendix

```
import numpy as np
from scipy.stats import f, norm, gaussian_kde
import matplotlib.pyplot as plt
import scipy

n = 50
k = 4
num_sim = 100000

# Generate X matrix
X = np.hstack([np.ones((n, 1)), np.random.normal(loc = 0, scale = 1, size=(n
                                         , k-1))])

def theoretical(sigma2, beta, DEBUG = False):
  # Creating empty arrays for R^2 and F-statistic values
  r2_arr = []
  f_arr = []

  for i in range(num_sim):

    #Generating error terms for different sigma2=1,2,4
    e = np.random.normal(loc = 0, scale =sigma2, size=n)

    #This is from Wikipedia
    beta_hat = beta + np.linalg.inv(X.T @ X) @ X.T @ e

    #Computing true y
    y = X @ beta + e

    # Generating Y_hat and residuals for sigma^2
    y_hat = X @ beta_hat

    #Computing the R^2 statistic as in the book
    ess = np.sum( (y_hat - np.mean(y))**2 )
    tss = np.sum( (y - y_hat)**2 ) + ess
    r2 = ess/tss

    if DEBUG:
      print(r2)

    # Store R-squared and sigma^2 values
    r2_arr.append(r2)

    if DEBUG:
      print(r2_arr)

    # Calculating F-statistic
    dfn = k-1
    dfd = n-k

    fstat = dfd/dfn*(r2/(1-r2))

    # Store F-statistic values
    f_arr.append(fstat)

  return r2_arr, f_arr

#plot function for F-statistic
def plot_f(f_arr, beta, sigma2):
    if beta == 0:
        # Plotting F-statistic values
        plt.hist(f_arr, bins=100, density=True, alpha=0.5, stacked = True,
                                                label='Empirical F-statistic')

        p = scipy.stats.f.pdf(np.linspace(0,10, 500), k-1, n-k)
        plt.plot(np.linspace(0,10, 500), p, 'b', label='Theoretical F-
                                                statistic')

        #Kernel density
        kde = gaussian_kde(f_arr)
        dist_space = np.linspace(0, 10, 500)
        plt.plot(dist_space, kde(dist_space), color='red', label='Kernel
                                                density for empirical F-
                                                statistic')

        plt.title('F-statistic values for $\sigma^2$ = {}'.format(sigma2))
        plt.xlim(0,10)
        plt.legend()
        plt.show()

    #plot for beta = (1,1,1,1)
    else:
        # Plotting F-statistic values
        plt.hist(f_arr, bins=100, density=True, alpha=0.5, stacked = True,
                                                color = 'cyan', label='
                                                Empirical F-statistic')

        #Kernel density
        kde = gaussian_kde(f_arr)
        dist_space = np.linspace(0, 150, 1000)
        plt.plot(dist_space, kde(dist_space), color='red', label='Kernel
                                                density for empirical F-
                                                statistic')

        plt.title('F-statistic values for $\sigma^2$ = {}'.format(sigma2))
        plt.xlim(0,150)
        plt.legend(loc = 'upper right')
        plt.show()
    return

#plotting R^2 for beta = (0,0,0,0)
plot_r2(theoretical(1, np.zeros(k))[0],0, 1)
plot_r2(theoretical(2, np.zeros(k))[0],0, 2)
plot_r2(theoretical(4, np.zeros(k))[0],0, 4)


#Plotting F-statistic for beta = (0,0,0,0)
plot_f(theoretical(1, np.zeros(k))[1],0, 1)
plot_f(theoretical(2, np.zeros(k))[1],0, 2)
plot_f(theoretical(4, np.zeros(k))[1],0, 4)


#plotting R^2 for beta = (1,1,1,1)
plot_r2(theoretical(1, np.ones(k))[0],1, 1)
plot_r2(theoretical(2, np.ones(k))[0],1, 2)
plot_r2(theoretical(4, np.ones(k))[0],1, 4)


#Plotting F-statistic for beta = (1,1,1,1)
plot_f(theoretical(1, np.ones(k))[1],1, 1)
plot_f(theoretical(2, np.ones(k))[1],1, 2)
plot_f(theoretical(4, np.ones(k))[1],1, 4)
```

Code Section 1: Comparison of F-statistic and $R^2$-statistic computed via simulation and its theoretical counterpart. We do experiments for $\sigma = 1, 2$ and 4 and $\beta = 0$ and 1.

```
import numpy as np
import matplotlib.pyplot as plt

T = 100
sigma2 = 2
num_sims = 10000


# Iterating over different number of regressors (k)
r2_array = np.zeros((num_sims, 9))
for k in range(2, 11):
    for i in range(num_sims):

        # Generating X matrix for each simulation
        X_sim = np.random.normal(loc = 0, scale = 1, size=(T, k))
        X_sim[:,0] = 1

        # Error term
        e = np.random.normal(loc = 0, scale=sigma2, size=T)

        # Calculating the two betas
        beta = np.zeros(k)
        beta_hat = np.linalg.inv(X_sim.T @ X_sim) @ X_sim.T @ e

        # Calculating true and fitted y
        y = X_sim @ beta + e
        y_hat = X_sim @ beta_hat

        # Estimate R^2
        ess = np.sum( (y_hat - np.mean(y))**2 )
        tss = np.sum( (y - y_hat)**2 ) + ess
        r2 = ess/tss

        # Saving results
        r2_array[i, k-2] = r2

# Generating boxplots of R^2
plt.boxplot(r2_array, labels=list(range(2, 11)))
plt.xlabel('Number of regressors')
plt.ylabel('R-squared')
plt.title('Distribution of $R^2$ for different number of regressors')
plt.show()
```

Code Section 2: Simulation study for the assessment of the $R^2$-statistic for an increasing number of random regressors.

```
def make_cov(a_est, T):
    T = T+1

    #init Identity matrix
    cov = np.eye(T)

    for row in range(T):
        for column in range(T):
            if column != row:
                #if the the element is not on the diagonal, then the power
                # is the difference between the current row and current
                                                               column
                cov[row,column] = a_est ** np.abs((column-row)) / (1-a_est
                                                               ** 2)
            else:
                #if row == column, the element is on the main diagonal, i.e.
                                                                1
                continue
    return cov
```

Code Section 3: Function to create the covariance matrix for the error terms. The function is implemented using dynamic programming.

```
#set initial variables
DEBUG = False
T = 50
sim = 10000
sigma2 = 1

#generate matrix with 4 iid samples from N(0,1)
X = np.random.normal(4,1,(T+1,4))
X = np.concatenate((np.ones((T+1,1)),X),axis=1)

#init lists where the results will be stored
beta0 = list()
beta1 = list()
beta2 = list()
beta3 = list()
beta4 = list()
as_ = list()
variances = list()

for simulation in range(sim):

    print(simulation)

    #init parameters for the simulation
    estimate_beta = np.zeros(5)
    estimate_a = 0.6
    Y = np.zeros(T+1)
    U = np.random.normal(0, sigma2, T)

    #simulate the AR(1) process
    for t in range(T):
        if t == 0:
```

```
        #If a epsilon is in (-1, 1), then epsilon_0 is assumed to be a
                                                realization from its
                                                unconditional distribution
                                                , N(0, sigma^2 / (1 - a^2)
                                                ),
        Y[t] = X[t]@estimate_beta + estimate_a*np.random.normal(0,
                                                sigma2/(1-estimate_a**2))
                                                + U[t]
    else:
        e_t_1 = Y[t-1] - X[t-1]@estimate_beta
        Y[t] = X[t]@estimate_beta + estimate_a*e_t_1 + U[t]

if DEBUG:
    print(f"Y: {Y}")

#Make matrices to compute a
D_t_1 = np.concatenate( ( np.eye(T), np.zeros((T, 1)) ), axis = 1)
D_t = np.concatenate( ( np.zeros((T, 1)), np.eye(T) ), axis = 1)

#compute the initial beta estimator (OLS)
beta_est = np.linalg.inv(X.T@X)@X.T@Y

if DEBUG:
    print(f"beta_est_LS: {beta_est}")

#derive the error terms
e_t = Y - X@beta_est

#estimate a based on the error terms
upper = (e_t.T@D_t_1.T@D_t@e_t)
lower = (e_t.T@D_t_1.T@D_t_1@e_t)
a_est = upper / lower

if DEBUG:
    print(f"upper: {upper}")
    print(f"lower: {lower}")
    print(f"a_est_LS: {a_est}")

#estimate variance
var_est = np.sum(((e_t.T@D_t.T) - a_est*(e_t.T@D_t_1.T))**2) / (T - 5)


#init variables for the while loop
estimate_beta = beta_est
estimate_a = a_est
iterator = 0
no_convergence = True
while no_convergence:

    #compute covariance matrix accroding to 4.20 (Paolella, 2018)
    cov = make_cov(estimate_a, T)
    inv_cov = np.linalg.inv(cov)

    #compute beta from Generalized Least Squares (GLS)
    beta_est = np.linalg.inv(X.T@inv_cov@X)@X.T@inv_cov@Y

    #once again, derive the error terms
    e_t = Y - X@beta_est
```

```
        #compute a based on the error terms
        upper = e_t.T@D_t_1.T@D_t@e_t
        lower = e_t.T@D_t_1.T@D_t_1@e_t
        a_est = upper / lower

        if ( (all( np.round(beta_est, 5) == np.round(estimate_beta, 5) ) )
                                                      and
             ( np.round(a_est, 5) == np.round(estimate_a, 5) ) or
             (iterator > 500)):

            #break the while loop since the algorithm has converged
            no_convergence = False

            if iterator > 500:
                #if there is no convergence, then the algorithm will be
                                                              restarted
                print("no convergence")
                iterator = 0
                continue
            else:
                beta0.append(beta_est[0])
                beta1.append(beta_est[1])
                beta2.append(beta_est[2])
                beta3.append(beta_est[3])
                beta4.append(beta_est[4])
                as_.append(a_est)
                variances.append(var_est)

            if DEBUG:
                print("--------------------- Final result
                                                              ---------------------"
                                                              )
                print(f"beta_est: {beta_est} & estimate_beta: {estimate_beta
                                                              }")
                print(f"a_est: {a_est} & estimate_a: {estimate_a}")
                print(f"var_est: {var_est} & true_sigma: {sigma2}")

        else:
            #update the estimates
            estimate_beta = beta_est
            estimate_a = a_est
            estimate_var = var_est

            iterator += 1

            if DEBUG:
                print(f"beta_est: {beta_est}")
                print(f"estimate_a: {a_est}")
                print("----------------- Iteration has ended
                                                              -----------------")
```

Code Section 4: The simulation study of section 3.