Question 1

Generate 500 observations from an AR(1) process Y_t with $\mathbb{E}[Y_t] = 0$, $\phi = 0.4$ and the variance of the white-noise forcing term $\sigma^2_\varepsilon = 0.2$ using the two methods below (hint: the random number generator for i.i.d. normal is `randn`).

a. A `for` loop using the recursive structure of the AR(1).

Solution. The exercise asks to simulate 500 observations from an AR(1) process:

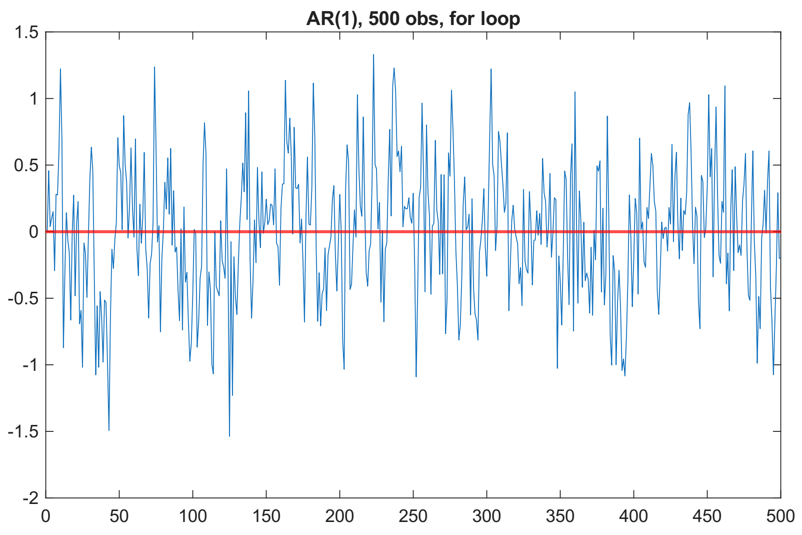
$$Y_t = \phi Y_{t-1} + \epsilon_t \qquad \epsilon_t \sim N(0, \sigma^2_\epsilon) \tag{1}$$

such that $\mathbb{E}[Y_t] = 0$, $\phi = 0.4$ and the variance of the white-noise forcing term $\sigma^2_\varepsilon = 0.2$.

Employing the random number generator `randn`, which generates by default a vector such that $Z \sim N(0, 1)$, it is enough to initialize σ^2_ϵ as required and compute `sqrt` to generate the white-noise forcing term distribution using the property of linear transformations of standard normal variables: if $Z \sim N(0, 1)$, then $\sigma Z \sim N(0, \sigma^2)$.

```
%AR1: y_t = \phi y_{t-1} + \epsilon_t
T = 500;
phi = 0.4;
sigma2 = 0.2;
epsilon = sqrt(sigma2)*randn(T,1); %\sigmaN(0,1) = N(0,\sigma^2)

%for loop
y=zeros(T,1);
y(1)=epsilon(1);
for t = 2:T
    y(t) = phi*y(t-1) + epsilon(t);
end
plot(y)
title('AR(1), 500 obs, for loop')
```

The resulting process has the following representation.



□

b. Using the function `filter`.

Solution. The function `filter` employs the polynomial structure of the AR(1) and the lag operator L. Any AR(p) process, indeed, can be written as:

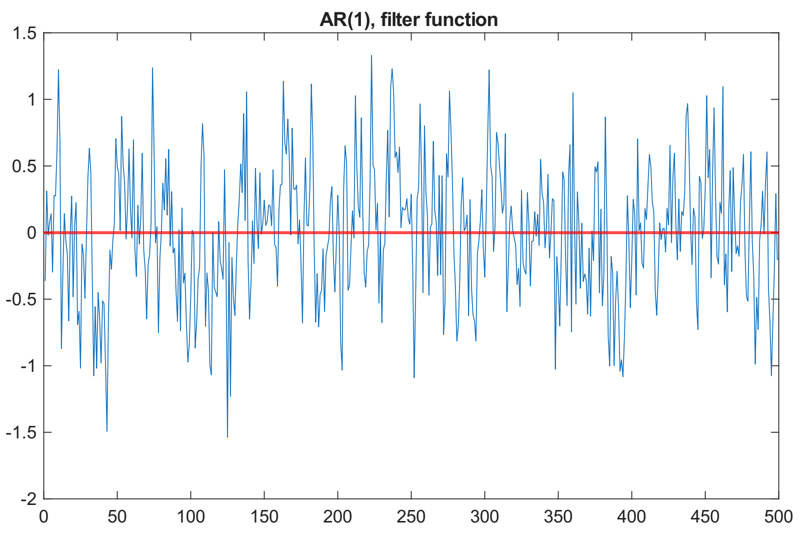
$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + ... + \phi_p Y_{t-p} + \epsilon_t \implies (1 - \phi(L))Y_t = \epsilon_t \tag{2}$$

In the case of the AR(1), the polynomial is simply ϕL and, given $\phi = 0.4$, $|\phi| < 1$. This implies that the process is stationary and well-defined and can be rewritten as:

$$(1 - \phi L)Y_t = \epsilon_t \implies Y_t = (1 - \phi L)^{-1}\epsilon_t \tag{3}$$

```
%AR(1): (1-\phiL)y_t = \epsilon_t
y_filter = filter(1,[1 -phi],epsilon);
plot(y_filter)
title('AR(1), filter function')
```

The plot retrieves the following representation:



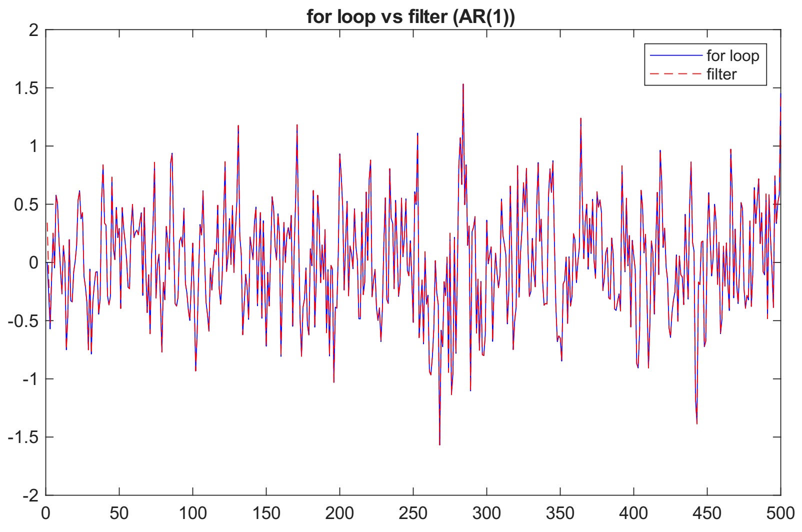
□

c. Check that when the forcing variables are the same, the output of the two approaches is the same (be careful with the starting conditions and with the random number generator).

Solution. In order to verify that the two approaches retrieve the same process, we compute the max of the absolute value of the difference between the two. We should expect it to be exactly equal to zero (this is also the reason which led us, when running the `for` loop, we initialise $y(1) = \epsilon(1)$, in order to maintain coherence with the `filter` function).

```
%confront
diff_val = max(abs(y - y_filter));
fprintf('max difference: %.2e\n', diff_val);

plot(y,'b'); hold on;
plot(y_filter, '--r');
legend('for loop', 'filter');
title('for loop vs filter (AR(1))');
hold off;
```

The max difference is indeed zero. We provide also a representation of the two series.



□

Question 2

Generate data from an AR(1) with $\phi = 0.6$, $\sigma^2_\varepsilon = 0.4$ and $\mathbb{E}[Y_t] = 3$. Set the starting condition of your simulation to 20. What happens if the starting condition you choose is far from the unconditional mean of the process? What would you do in order to make sure that the sample path is a "proper" realization of the stationary process you want to simulate from? You can use either `for` or `filter`.

Solution. The MA(∞) representation of the AR(1) without the constant:

$$Y_t = \phi Y_{t-1} + \epsilon_t \implies Y_t = \sum_{j=0}^{\infty} \phi^j \epsilon_{t-j} \tag{4}$$

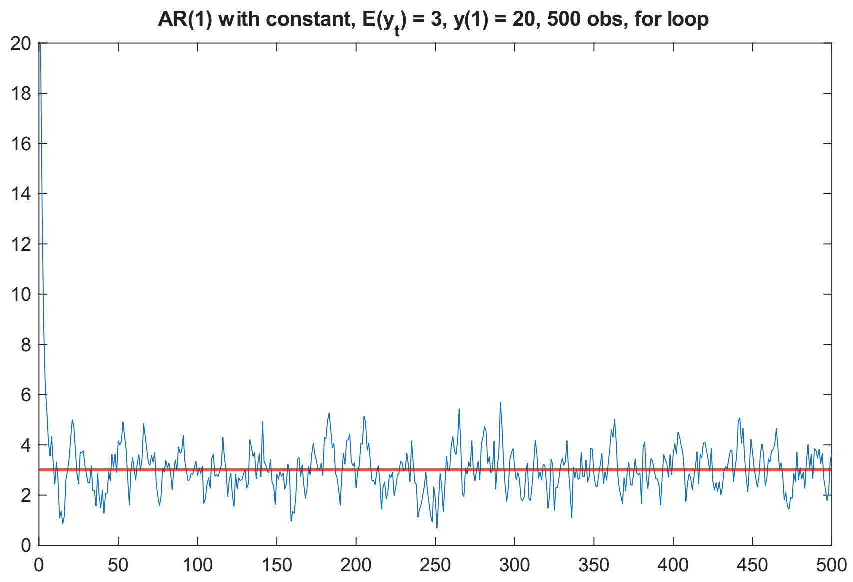
Allows us to see clearly that $\mathbb{E}(Y_t) = 0$. Therefore, to have $\mathbb{E}(Y_t) = 3$ as required, we should rewrite the AR(1) model adding a constant c:

$$Y_t = c + \phi Y_{t-1} + \epsilon_t \implies Y_t = \frac{c}{1-\phi} + \sum_{j=0}^{\infty} \phi^j \epsilon_{t-j} \tag{5}$$

To compute the necessary value of c such that $\mathbb{E}(Y_t) = 3$, it is enough to plug in the expected value of the AR(1) with constant the given value of ϕ to retrieve $c = 1.2$.

```
%AR1: y_t = c+\phi y_{t-1} + \epsilon_t
% E(y_t) = \frac{c}{1-\phi} = 3
c = 1.2;

phi_b = 0.6;
sigma2_b = 0.4;
epsilon_b = sqrt(sigma2_b)*randn(T,1); %\sigmaN(0,1) = N(0,\sigma^2)

%for loop
y_b=zeros(T,1);
y_b(1)=20;
for t = 2:T
    y_b(t) = c+phi_b*y_b(t-1) + epsilon_b(t);
```

```
end
plot(y_b)
title('AR(1) with constant, E(y_t) = 3, y(1) = 20, 500 obs, for loop')
```



As shown by the graph, setting the starting condition far from the unconditional mean of the process leads the serie to accelerate towards the true expected value. As a matter of fact, computing the expected value of the process given Y_1 (subtracting $\mu = \frac{c}{1-\phi} = 3$ on both sides):

$$Y_t - \mu = c + \phi Y_{t-1} + \epsilon_t - \frac{c}{1-\phi} \implies y_t - \mu = \phi(Y_{t-1} - \mu) + \epsilon_t$$

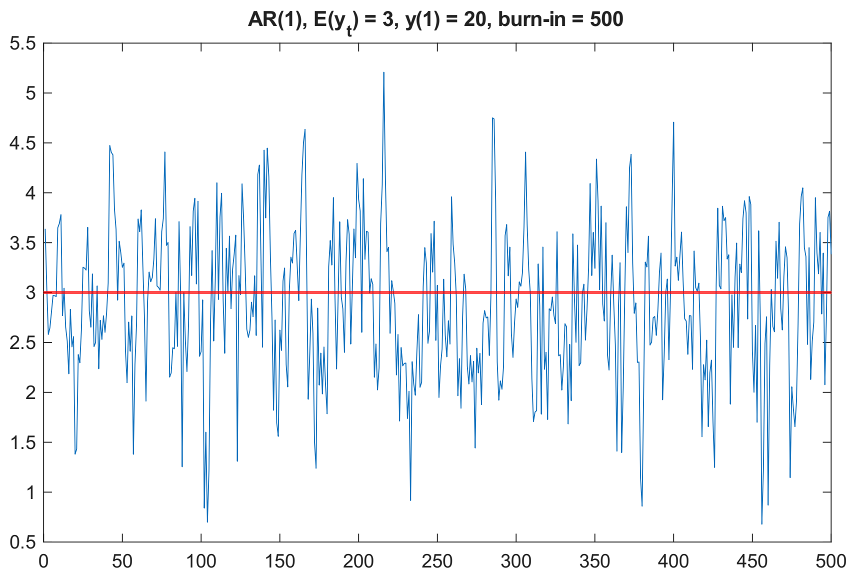
$$Y_t - \mu = \phi^{t-1}(Y_1 - \mu) + \sum_{j=0}^{t-2} \phi^j \epsilon_{t-j}$$

$$\mathbb{E}[Y_t - \mu \mid Y_1] = \mathbb{E}[\phi^{t-1}(Y_1 - \mu) \mid Y_1] + 0$$
$$\mathbb{E}[Y_t \mid Y_1] = \mu + \phi^{t-1}(Y_1 - \mu)$$

Therefore, the distance of the process from the unconditional mean decays geometrically.

To obtain a sample path which is a "proper" realisation of the desired stationary process we want to simulate from we should insert a burn-in sample period, which we fix to the first 500 observations, followed by the path we effectively maintain. In this way, we discard the initial transitory part and we focus uniquely on the targeted process. The results, as can be observed, oscillate coherently around the true unconditional mean, i.e. 3.

```
% burn-in
T_burn = 500;             % burn-in period

eps_long = sqrt(sigma2_b)*randn(T_burn+T,1);

y_long = zeros(T_burn+T,1);
y_long(1) = 20;
for t = 2:(T_burn+T)
    y_long(t) = c + phi_b*y_long(t-1) + eps_long(t);
end

y_burned = y_long(T_burn+1:end);
plot(y_burned)
title('AR(1), E(y_t) = 3, y(1) = 20, burn-in = 500')
```



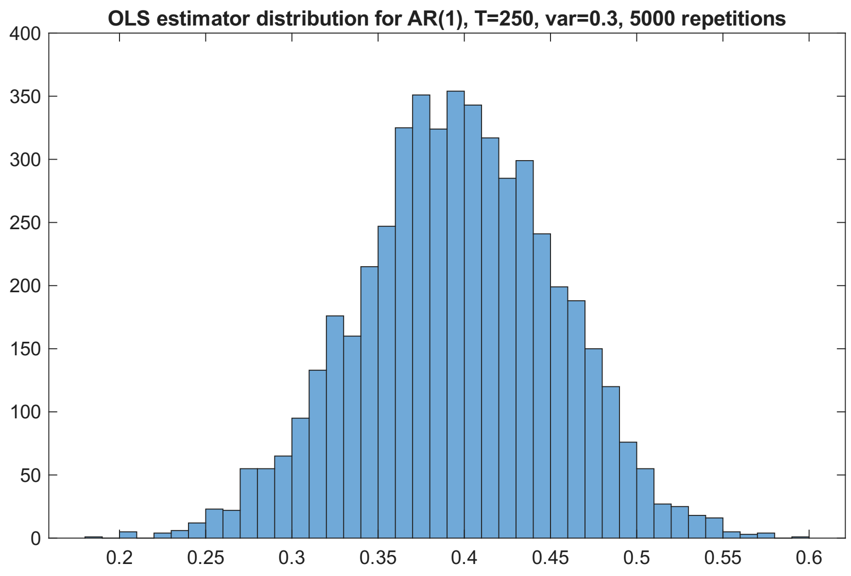
□

Question 3

Compute the empirical distribution of the OLS estimator in the case of an AR(1) with $\phi = 0.4$ and $T = 250$ (you are free to choose the variance of the innovation). Construct a t-test for the null hypothesis $H_0: \ \phi = 0$, against a two-sided alternative $H_1: \ \phi \neq 0$. How often do you reject H_0 at the 95% confidence level when $T = 250$?

Solution. We have simulated a Monte-Carlo experiment with 5000 repetitions of the same `for` loop we have already implemented in Exercises 1 and 2 in order to simulate the AR(1) process. Moreover, to initialise the process and avoid distorting effects of the starting condition, we inserted a burn-in of 200 observations. For each repetition of the loop, we stored the coefficient β in a vector of dimension 5000 and we plotted the distribution of the values in the vector through an histogram.

```
% OLS estimator: beta = \frac{Cov(y_t,y_{t-1})}{V(y_{t-1}


)}
sigma2_c = 0.3;
phi_c = 0.4;
T_c = 250;
T_burn = 200;
B = 5000;

beta = zeros(B,1);
tstats = zeros(B,1);

for b = 1:B
    epsilon_c  = sqrt(sigma2_c)*randn(T_c + T_burn,1);
    y_c = zeros(T_c+T_burn,1);
    y_c(1) = 0;
    for t = 2:(T_c + T_burn)
        y_c(t) = phi_c*y_c(t-1) + epsilon_c(t);
    end
    y_c = y_c(T_burn+1:end);              %discard the burn-in
    Y  = y_c(2:end);
    X  = y_c(1:end-1);

    beta(b) = (X'*Y)/(X'*X);
    u = Y - beta(b)*X;                    % residuals
    RSS = u' * u;
    df  = (T_c - 1) - 1;
    sigma2_hat = RSS / df;
    se_beta = sqrt( sigma2_hat / (X' * X) );
    tstat_b = beta(b) / se_beta;

    tstats(b) = tstat_b;

end
histogram (beta)
title ('OLS estimator distribution for AR(1), T=250, var=0.3, 5000 repetitions'
    )
```



We then proceeded to construct the t-test for the null hypothesis that $\phi = 0$, first computing the residual sum of squares of each regression in order to calculate the estimated variance (and therefore the standard error) of the OLS estimator. This was useful to construct the `t-stat`:

$$t = \frac{\hat{\phi} - 0}{\text{se}(\hat{\phi})} \qquad \text{se}(\hat{\phi}) = \sqrt{\frac{\hat{\sigma}^2}{(X'X)}} \tag{6}$$

Defining the critical region of the two-tail test at 5%, the rejection rate of the test and mean, standard deviation and bias of $\hat{\phi}$ can be found in the following table.

Statistic	Value
Rejection rate @95% (T=250)	1.0000
Mean($\hat{\phi}$)	0.3971
Std($\hat{\phi}$)	0.0575
Bias($\hat{\phi}$)	−0.0029

Table 3-1: Simulation results for $\hat{\phi}$ estimates.

```
crit = 1.96;             % 95% threshold, two-tail
reject = abs(tstats) > crit;
rejection_rate = mean(reject);        % rejection frequence (in [0,1])
fprintf('Rejection rate @95%% (T=250): %.3f\n', rejection_rate);

```

```
% statistics
mean_beta = mean(beta);
std_beta  = std(beta);
bias_beta = mean_beta - phi_c;    % bias

fprintf('Mean(phi ) = %.4f\n', mean_beta);
fprintf('Std(phi )  = %.4f\n', std_beta);
fprintf('Bias(phi ) = %.4f\n', bias_beta);
```

□

Question 4

Compute the empirical distribution of the OLS estimator in the case of an AR(1) with $\phi = 0.9$ and $T \in \{50, 100, 200, 1000\}$. For each T, do 1000 simulations and plot the distribution. How is the distribution changing with T?

Solution. We estimate the autoregressive coefficient ϕ in a simple AR(1) model and study the behavior of its OLS estimator across different sample sizes.

The data generating process (DGP) is defined as:

$$x_t = \phi x_{t-1} + \varepsilon_t, \qquad \varepsilon_t \sim \text{iid } \mathcal{N}(0, \sigma^2), \quad (\phi, \sigma^2) = (0.9,\, 0.3). \tag{7}$$

The autoregressive coefficient $\phi = 0.9$ ensures a high degree of persistence while still satisfying the condition $|\phi| < 1$, which guarantees covariance stationarity and the existence of a finite unconditional variance. The innovation variance is set to $\sigma^2 = 0.3$, corresponding to a moderate level of stochastic noise in the process.

The OLS estimator of ϕ is given by:

$$\hat{\phi} = \frac{\sum_{t=2}^{T} x_t x_{t-1}}{\sum_{t=2}^{T} x_{t-1}^2}. \tag{8}$$

From asymptotic theory, under standard assumptions for a stationary AR(1) process, the OLS estimator satisfies:

$$\hat{\phi} \xrightarrow{p} \phi, \qquad \sqrt{T}(\hat{\phi} - \phi) \xrightarrow{d} \mathcal{N}\left(0,\, 1 - \phi^2\right),$$

which implies that it is both *consistent* and *asymptotically normal.*

To confirm these asymptotic properties empirically, we study the finite-sample distribution of $\hat{\phi}$ through Monte Carlo simulation. For each sample size ($T = 50, 100, 200, 1000$), we generate 1000 independent realizations of the process and compute the corresponding OLS estimates of ϕ. Each simulated series includes a burn-in of 200 observations to mitigate initialization effects, which are discarded before estimation. Figure 4-1 displays the empirical density plots of the estimated coefficients.

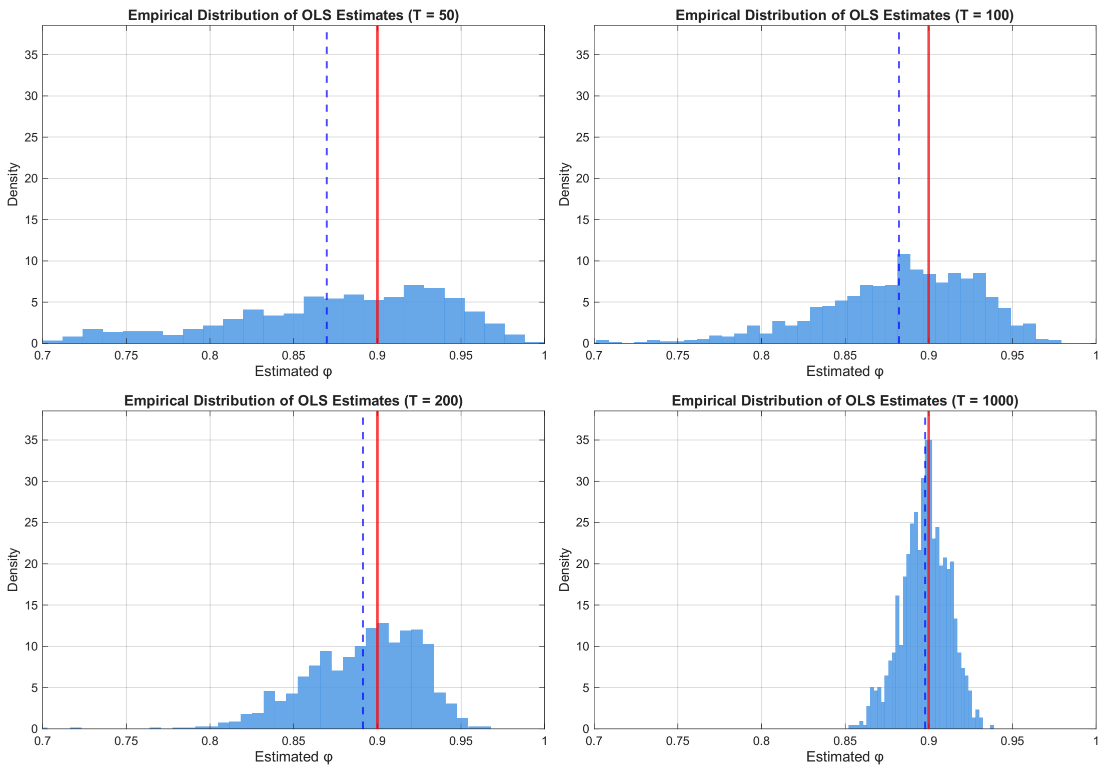


Figure 4-1: Empirical distributions of the OLS estimator $\hat{\phi}$ for different sample sizes ($T = 50, 100, 200, 1000$). The red line marks the true parameter $\phi = 0.9$, while the blue dashed line shows the sample mean of the estimates.

As T increases, the distribution becomes narrower and more centered around the true parameter. For $T = 50$, the estimates are highly dispersed and the bias is visible; for $T = 100$ and $T = 200$, dispersion decreases noticeably; and for $T = 1000$, the estimates cluster tightly around 0.9. This confirms the consistency of the OLS estimator in the AR(1) model — larger samples yield lower variance and smaller bias. The table below summarizes the main results across different sample sizes:

T	True ϕ	Mean($\hat{\phi}$)	Std($\hat{\phi}$)	Bias
50	0.900	0.872	0.072	-0.028
100	0.900	0.885	0.051	-0.015
200	0.900	0.893	0.036	-0.007
1000	0.900	0.899	0.012	-0.001

Table 4-2: Empirical mean, standard deviation, and bias of the OLS estimator $\hat{\phi}$ across 1000 simulations.

These results highlight that the OLS estimator becomes increasingly accurate as the sample size grows, both in terms of mean and dispersion, in line with the theoretical properties derived above.

```matlab
clear; clc; close all;
rng(123);  % set seed for reproducibility

phi_true = 0.9;
T_values = [50, 100, 200, 1000];
nSim     = 1000;
sigma_2  = 0.3;
```

```
burnin    = 200;

phi_hat_all = cell(length(T_values),1);
mean_vals = zeros(length(T_values),1);
std_vals   = zeros(length(T_values),1);
delta_vals = zeros(length(T_values),1);

for k = 1:length(T_values)
    T = T_values(k);
    N = T + burnin;
    phi_estimates = zeros(nSim,1);

    for s = 1:nSim
        x = zeros(N,1);
        eps = sqrt(sigma_2) * randn(N,1);
        for t = 2:N
            x(t) = phi_true * x(t-1) + eps(t);
        end
        x = x(burnin+1:end);
        y = x(2:end);
        X = x(1:end-1);
        phi_est = (X' * y) / (X' * X);
        phi_estimates(s) = phi_est;
    end

    phi_hat_all{k} = phi_estimates;
    mean_vals(k) = mean(phi_estimates);
    std_vals(k)   = std(phi_estimates);
    delta_vals(k) = mean_vals(k) - phi_true;
end

x_limits = [0.7, 1.0];
max_density = 0;
for k = 1:length(T_values)
    [counts, edges] = histcounts(phi_hat_all{k}, 40, 'Normalization', 'pdf');
    max_density = max(max_density, max(counts));
end
```

□

Question 5

Compute the empirical distribution of the OLS estimator in the regression $x_t = ax_{t-1} + v_t$ when the true DGP for x_t is MA(1) with $\theta = 0.6$ for $T \in \{50, 100, 200, 1000\}$. What is the mean of the distributions? Does the mean converge to anything as $T \to \infty$? Discuss.

Solution. We estimate the autoregressive coefficient a in a misspecified AR(1) model when the true data-generating process (DGP) follows a moving average of order one, MA(1). The goal is to analyze how the OLS estimator behaves under model misspecification and whether it converges to the theoretical pseudo-true value.

The true DGP is defined as:

$$x_t = \varepsilon_t + \theta\varepsilon_{t-1}, \qquad \varepsilon_t \sim \text{iid } \mathcal{N}(0, \sigma^2), \quad (\theta, \sigma^2) = (0.6,\ 0.3). \tag{9}$$

The moving-average parameter $\theta = 0.6$ generates a moderate level of short-run serial correlation in the data, while the innovation variance $\sigma^2 = 0.3$ implies a moderate amount of stochastic noise in the process.

Even though the true model is MA(1), we estimate an AR(1) by OLS according to

$$x_t = ax_{t-1} + v_t, \tag{10}$$

where a is the autoregressive coefficient obtained from the linear projection of x_t on x_{t-1}. Under this misspecification, the OLS estimator converges to the pseudo-true value

$$a^* = \frac{\text{Cov}(x_t, x_{t-1})}{\text{Var}(x_{t-1})} = \frac{\theta}{1+\theta^2}, \quad \text{which equals} \quad a^* = \frac{0.6}{(1+0.6^2)} \approx 0.441$$

To confirm this theoretical property empirically, we study the finite-sample distribution of $\hat{a}$ through Monte Carlo simulation. For each sample size ($T = 50, 100, 200, 1000$), we generate 1000 independent realizations of the process and compute the corresponding OLS estimates of a. Each simulated series includes a burn-in of 200 observations to mitigate initialization effects, which are discarded before estimation. Figure 5-2 displays the empirical density plots of the estimated coefficients.

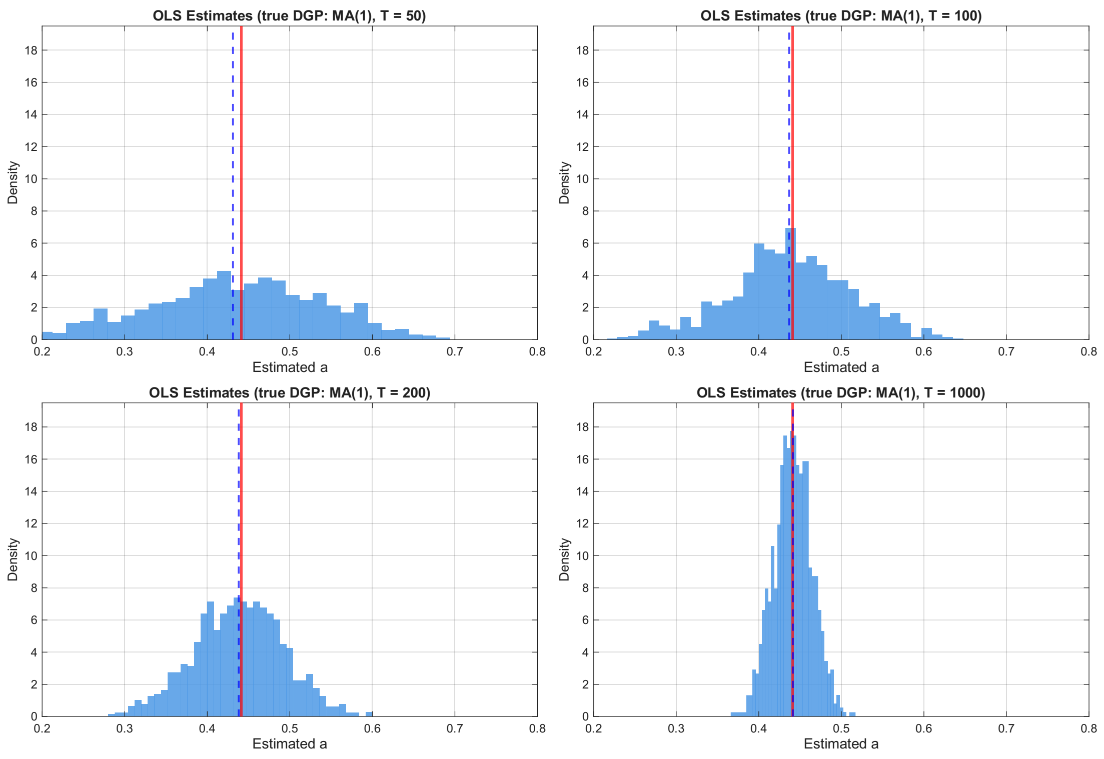


Figure 5-2: Empirical distributions of the OLS estimator $\hat{a}$ when the true DGP is MA(1) with $\theta = 0.6$. The red line marks the pseudo-true value $a^* = \theta/(1+\theta^2)$, while the blue dashed line indicates the mean of the simulated OLS estimates.

As the sample size increases, the distributions become narrower and more centered around the pseudo-true value. For small samples, the estimates are more dispersed, but as T grows the variance shrinks and the mean approaches a^*. This confirms that, even under model misspecification, the OLS estimator converges to the pseudo-true value rather than to the true θ as $T \to \infty$.

T	True a^*	Mean($\hat{a}$)	Std($\hat{a}$)	Bias
50	0.441	0.432	0.068	-0.009
100	0.441	0.437	0.049	-0.004
200	0.441	0.440	0.034	-0.001
1000	0.441	0.441	0.011	≈ 0

Table 5-3: Empirical mean, standard deviation, and bias of the OLS estimator $\hat{a}$ across 1000 simulations.

The results clearly show that the OLS estimator becomes increasingly accurate as the sample size increases, with both its mean and dispersion converging to the theoretical pseudo-true value a^*.

□

```
clear; clc; close all;
rng(1);

theta_true = 0.6;
a_true = theta_true / (1 + theta_true^2);
T_values = [50, 100, 200, 1000];
nSim = 1000;
sigma_2 = 0.3;
burnin = 200;

a_hat_all = cell(length(T_values),1);
mean_vals = zeros(length(T_values),1);
std_vals  = zeros(length(T_values),1);
delta_vals = zeros(length(T_values),1);

for k = 1:length(T_values)
    T = T_values(k);
    N = T + burnin;
    estimates = zeros(nSim,1);

    for s = 1:nSim
        eps = sqrt(sigma_2) * randn(N+1,1);
        x_full = zeros(N,1);
        for t = 1:N
            x_full(t) = eps(t+1) + theta_true * eps(t);
        end
        x = x_full(burnin+1:end);
        y = x(2:end);
        X = x(1:end-1);
        a_est = (X' * y) / (X' * X);
        estimates(s) = a_est;
    end

    a_hat_all{k} = estimates;
    mean_vals(k) = mean(estimates);
    std_vals(k)  = std(estimates);
    delta_vals(k) = mean_vals(k) - a_true;
end

x_limits = [0.2, 0.8];
max_density = 0;
for k = 1:length(T_values)
    [counts, edges] = histcounts(a_hat_all{k}, 40, 'Normalization', 'pdf');
    max_density = max(max_density, max(counts));
end
```

Question 6

Write a function that generates T observations from an ARMA(p, q) using the `for`-loop approach. The function must have the following inputs:

(i) the number of observations;

(ii) the variance of the white noise ε_t

(iii) the coefficients of the AR and MA polynomials or the roots of the AR and MA polynomials (hint: you may find the `poly` and `roots` functions useful);

and as output the realizations of the ARMA(p,q) and the white noise ε_t.

Solution. The purpose of this exercise is to construct a MATLAB function that generates T observations from an ARMA(p,q) process using an explicit `for`-loop.

An ARMA(p,q) process can be represented in two equivalent ways: either through its *coefficients* (ϕ_i, θ_j) or through the *lag roots* $(\lambda_i^{AR}, \lambda_j^{MA})$ of its autoregressive and moving-average polynomials. The model is defined as

$$x_t = \sum_{i=1}^{p} \phi_i x_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}, \qquad \varepsilon_t \sim \mathcal{N}(0,\sigma^2),$$

which can be compactly expressed in terms of lag polynomials as

$$A(L)x_t = B(L)\varepsilon_t, \qquad A(L) = 1 - \sum_{i=1}^{p} \phi_i L^i, \qquad B(L) = 1 + \sum_{j=1}^{q} \theta_j L^j.$$

The process is stationary and invertible when all roots of $A(z) = 0$ and $B(z) = 0$ lie outside the unit circle. Alternatively, the same model can be parameterized in terms of lag roots:

$$A(L) = \prod_{i=1}^{p}(1 - \lambda_i^{AR} L), \qquad B(L) = \prod_{j=1}^{q}(1 + \lambda_j^{MA} L),$$

from which the coefficients are recovered by expanding the polynomials and collecting the powers of L:

$$\phi = -[\text{coefficients of } L, L^2, \ldots, L^p \text{ in } A(L)], \qquad \theta = [\text{coefficients of } L, L^2, \ldots, L^q \text{ in } B(L)].$$

These two representations are algebraically equivalent, and the choice between them depends on whether one prefers to work with direct coefficients or with the underlying lag roots that determine the stationarity and invertibility properties of the process.

The implemented MATLAB function `generate_ARMA` reproduces this theoretical framework by generating T observations from an ARMA(p,q) process through explicit recursion. The user specifies the number of observations T, the innovation variance σ^2, and the vectors of AR and MA parameters, which may represent either coefficients or lag roots. To distinguish between these two cases, the function includes a logical input argument `isRoot` that takes the value `true` if the inputs correspond to lag roots, and `false` (the default) if they are coefficients. When `isRoot = true`, the function internally expands the polynomials $A(L)$ and $B(L)$ using the convolution operator and extracts the implied coefficients before performing the simulation. A burn-in period of 200 observations is added to remove dependence on initial conditions, and only the last T simulated observations are returned.

For greater rigor and code modularity, the function `generate_ARMA` was defined in a separate MATLAB file from the main script, allowing it to be reused, tested, and called independently without relying on the problem set code.

```matlab
function [x, eps] = generate_ARMA(T, sigma2, AR_input, MA_input, isRoot)
% ---------------------------------------------------------------
% Generates T observations from an ARMA(p,q) process
%
%   x_t = sum_{i=1}^p phi_i * x_{t-i} + eps_t
%         + sum_{j=1}^q theta_j * eps_{t-j}
%
% Inputs:
%   T          - number of observations
%   sigma2     - innovation variance
%   AR_input   - AR coefficients or roots
%   MA_input    - MA coefficients or roots
%   isRoot      - = true if inputs are roots, = false (default) if coefficients
%
% Outputs:
%   x    - simulated ARMA(p,q) series
%   eps - white noise innovations
% ---------------------------------------------------------------

if nargin < 5
    isRoot = false;  % default: interpret as coefficients
end

% --- Convert roots to coefficients if needed ---
if isRoot
    % AR polynomial: A(L) = prod_i (1 -  _i  L) = 1 -    L    - ... -  _pL ^p
    if ~isempty(AR_input)
        polyA = 1;
        for i = 1:length(AR_input)
            polyA = conv(polyA, [1, -AR_input(i)]);
        end
        ARcoeff = -polyA(2:end);
    else
        ARcoeff = [];
    end

    % MA polynomial: B(L) = prod_j (1 +  _j  L) = 1 +    L    + ... +  _qL ^q
    if ~isempty(MA_input)
        polyB = 1;
        for j = 1:length(MA_input)
            polyB = conv(polyB, [1, MA_input(j)]);
        end
        MAcoeff = polyB(2:end);
    else
        MAcoeff = [];
    end
else
    ARcoeff = AR_input(:)';   % ensure row vector
    MAcoeff = MA_input(:)';
end

% --- Orders and burn-in ---
p = length(ARcoeff);
q = length(MAcoeff);
burnin = 200;
N = T + burnin;

% --- Allocate and simulate ---
x_full   = zeros(N,1);
eps_full = sqrt(sigma2) * randn(N,1);

for t = 1:N
    % AR part
    for i = 1:p
        if t - i > 0
            x_full(t) = x_full(t) + ARcoeff(i) * x_full(t - i);
        end
    end
    % MA part
    for j = 1:q
        if t - j > 0
            x_full(t) = x_full(t) + MAcoeff(j) * eps_full(t - j);
        end
    end
    % Innovation
    x_full(t) = x_full(t) + eps_full(t);
end

% --- Drop burn-in period ---
x   = x_full(burnin+1:end);
eps = eps_full(burnin+1:end);

end
```

In the empirical implementation, we generate an ARMA(2,2) process with parameters

$$\phi = [0.7,\ -0.3], \qquad \theta = [0.5,\ 0.4], \qquad \sigma^2 = 1, \qquad T = 300.$$

The simulated series is displayed in Figure 6-3. The autoregressive polynomial

$$1 - 0.7L + 0.3L^2 = 0$$

has roots outside the unit circle, confirming stationarity. The moving-average terms introduce short-run dependence in the innovations, producing smoother oscillations compared to a pure AR process. The function was also tested under the root parameterization, confirming that the resulting time series are equivalent once the implied coefficients are computed from the polynomial expansion.

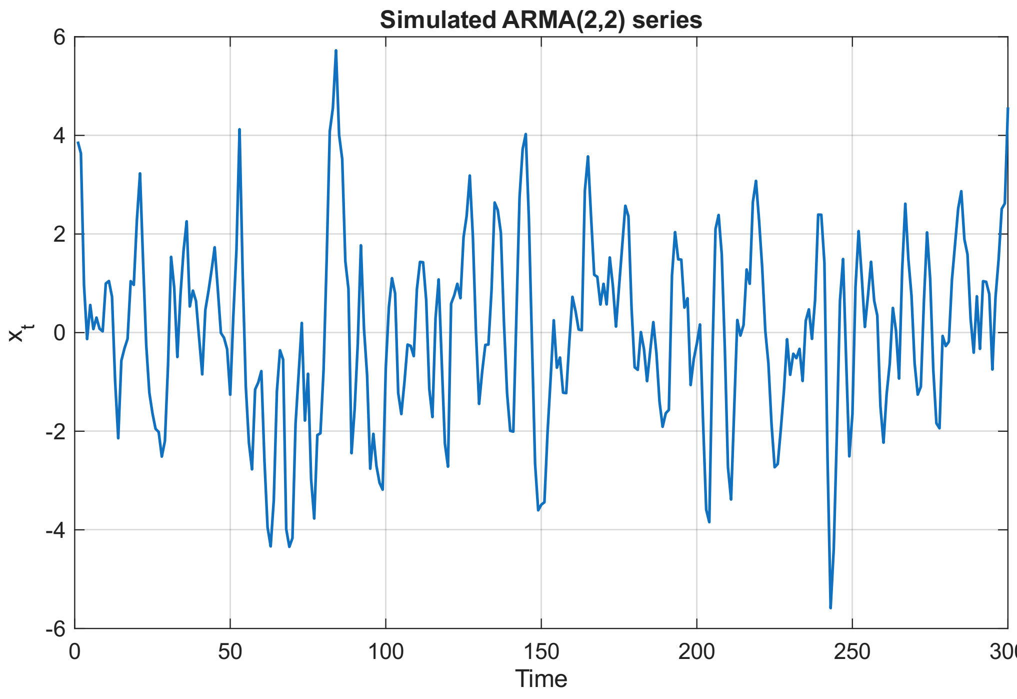


Figure 6-3: Simulated ARMA(2,2) process generated by the `generate_ARMA` function.

□

```
clear; clc; close all;
rng(123);  % for reproducibility

% --- Parameters ---
T        = 300;              % number of observations (after burn-in)
sigma2   = 1;                % white noise variance
phi      = [0.7, -0.3];      % AR(2) coefficients
theta    = [0.5, 0.4];       % MA(2) coefficients

% --- Generate ARMA(2,2) sample ---
[x, eps] = generate_ARMA(T, sigma2, phi, theta);
```