Problem 1

For kurtosis:

Assume the kurtosis function is not biased.
That is, the null hypothesis H0 is: the kurtosis function is not biased.
And the alternative hypothesis is: the kurtosis function is biased.
We set the threshold as 0.05 and conduct the student t test:
When the sample size is 100,000, and the number of samples is 100,
The p value is about 0.6291, which is larger than 0.05.
So, we fail to reject the null hypothesis, and the conclusion is that the kurtosis function is not biased.
However, as number of samples larger, the result changes.
When the sample size is 100, and the number of samples is 1000.
The p value is about 0.0149, which is smaller than 0.05.
So, now we could reject the null hypothesis, and we could say that the kurtosis function is biased, and the result is statistically significant.

For skewness:

Assume the skewness function is not biased.
That is, the null hypothesis H0 is: the skewness function is not biased.
And the alternative hypothesis is: the skewness function is biased.
Again, we set the threshold as 0.05 and conduct the student t test:
When the sample size is 100,000, and the number of samples is 1000,
The p value is about 0.9065, which is larger than 0.05.
So, we fail to reject the null hypothesis, and the conclusion is that the skewness function is not biased.
However, as number of samples larger, the result changes.
When the sample size is 10, and the number of samples is 1000.
The p value is about 0.0182, which is smaller than 0.05.
So, now we could reject the null hypothesis, and we could say that the skewness function is biased, and the result is statistically significant.

Here is the link to the code on Github:
https://raw.githubusercontent.com/SunYutongAmber/Quant-Risk-Analysis/main/Week02/project1_problem1.jl

Problem 2

Fit the data using OLS:
The coefficients for the linear regression is displayed as follows:

```
Coefficients:
──────────────────────────────────────────────────────────────────────
                Coef.  Std. Error     t  Pr(>|t|)  Lower 95%  Upper 95%
──────────────────────────────────────────────────────────────────────
(Intercept)  0.119836    0.121056  0.99    0.3246  -0.120396   0.360068
x            0.605205    0.124357  4.87    <1e-05   0.358423   0.851987
──────────────────────────────────────────────────────────────────────
```

, where beta0 is about 0.119836 and beta1 is about 0.605205.
Next, I test the normality of the error term using kurtosis and skewness with Hypothesis testing:

```
K2 = Zg1^2 + Zg2^2  #D'Agostino-Pearson statistic  14.146364750394484
X2 = K2  #approximation to chi-distribution  14.146364750394484

df = 2.  #degrees of freedom  2.0
P=1-ccdf(Chisq(df), X2)  0.9991524683507823
#println("P = $P ")

if P>alpha
    println("The error vector is normally distributed")
else
    println("The error vector is not normally distributed")
end  ✓

###################################################################
#Maximum Likelihood Estimation #MLE for Regression
df = CSV.read("problem2_data.csv", DataFrame)  100×2 DataFrame
x = df.x  100-element Vector{Float64}:
```

```
PROBLEMS 951   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS        Julia REPL (v1.9.3)

27.447554784922822
0.30015160770910915
3.5825858852297725
14.146364750394484
14.146364750394484
2.0
0.9991524683507823
The error vector is normally distributed
julia>
```

And we could find that the p-value is 0.99, which is larger than 0.05. So, we could conclude that the error vector is normally distributed.

Similar for using MLE for regression:
However, there is no built-in function for MLE in Julia, so we write:

```julia
function myll(s, b...)
    n = size(y,1)
    beta = collect(b)
    e = y - x.*beta
    s2 = s*s
    ll = -n/2 * log(s2 * 2 * π) - e'*e/(2*s2)
    return ll
end | myll (generic function with 2 methods)
```

And then we iterate for maximizing the likelihood using the optimizer function, and find the beta0 and beta1 for linear regression, which is 0.1196 and 0.60605191 respectively. We could find the regression model built with MLE is quite close to the model built with OLS.

```julia
#MLE Optimization problem
    mle = Model(Ipopt.Optimizer) | A JuMP Model
    set_silent(mle) | ✓

    @variable(mle, beta[i=1],start=0) | 1-dimensional DenseAxisArray{VariableRef,1,...} with index sets:
    @variable(mle, σ >= 0.0, start = 1.0) | σ

    register(mle,:ll,2,myll;autodiff=true) | ✓
    @NLobjective(
        mle,
        Max,
        ll(σ,beta...)
#########################
    ) | ✓
optimize!(mle) | ✓

println("Betas: ", value.(beta)) | ✓

b_hat = inv(x'*x)*x'*y | 0.6051912114646786
println("OLS: ", b_hat) | ✓
```

```
PROBLEMS 951   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS        Julia REPL (v1.9.3)

And data, a 1-element Vector{VariableRef}:
 beta[1]

σ



Betas: 1-dimensional DenseAxisArray{Float64,1,...} with index sets:
    Dimension 1, [1]
And data, a 1-element Vector{Float64}:
 0.6051912114615255

0.6051912114646786

OLS: 0.6051912114646786

julia>
```

Using the y_hat = beta0 + beta1*x

We could calculate the error vector and use skewness and kurtosis for Hypothesis testing to check the normality.

```
error = y - (x*0.6051912114646786 .+ 0.1196)   | 100-element Vector{Float64}:
mean(error)  | 0.0002361973796868777

# Perform the Hypothesis test to check normality of the error vector
# 'Agostino-Pearson's K2 test for assessing normality of data using skewness and kurtosis.
x = error | 100-element Vector{Float64}:
alpha = 0.05 | 0.05
```

Again, we could find that the p-value is about 0.99, which is larger than 0.05. So we could not reject the null hypothesis. And we could conclude that the error vector is normally distributed with statistical significance.

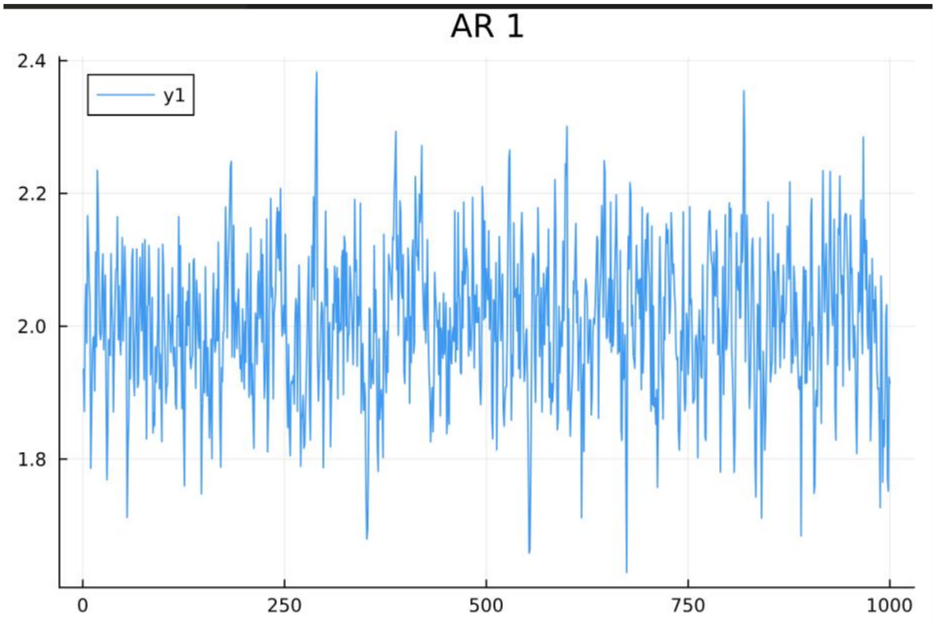
```

df = 2.  #degrees of freedom | 2.0
P=1-ccdf(Chisq(df), X2) | 0.9991524874979641
#println("P = $P ")
if P>alpha
    println("The error vector is normally distributed")
else
    println("The error vector is not normally distributed")
end | ✓
```

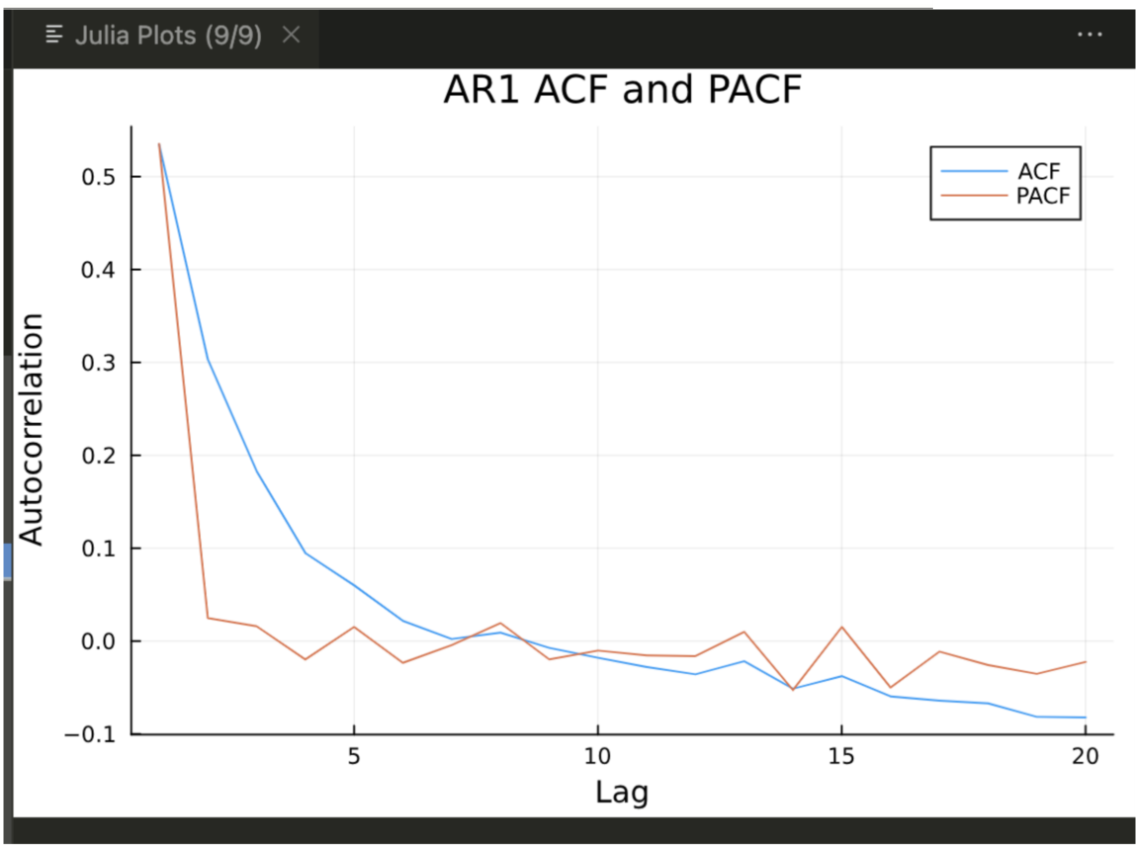
```
PROBLEMS 951   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

1.2771619842632693
27.447554784922822
0.30014975088204476
3.582601229588262
14.146409934307394
14.146409934307394
2.0
0.9991524874979641
The error vector is normally distributed
```

In general, OLS is better fit for the function. But the results are quite close.

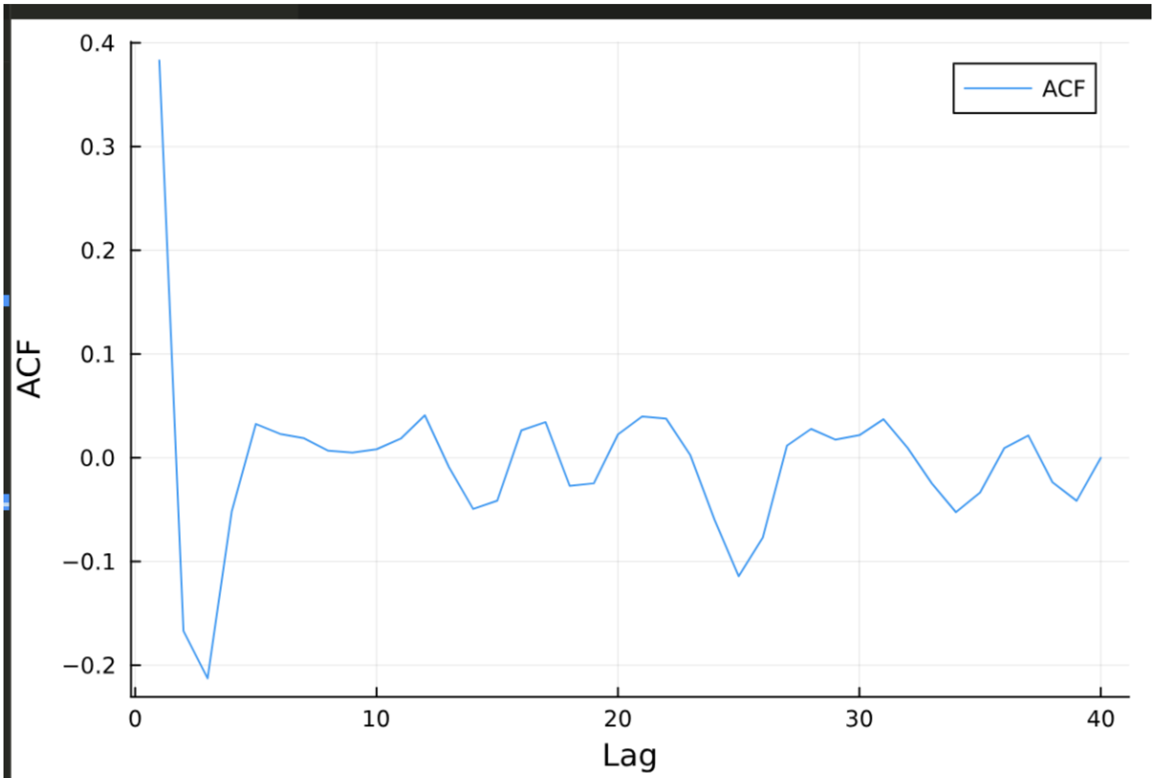
Problem 3

For AR1:

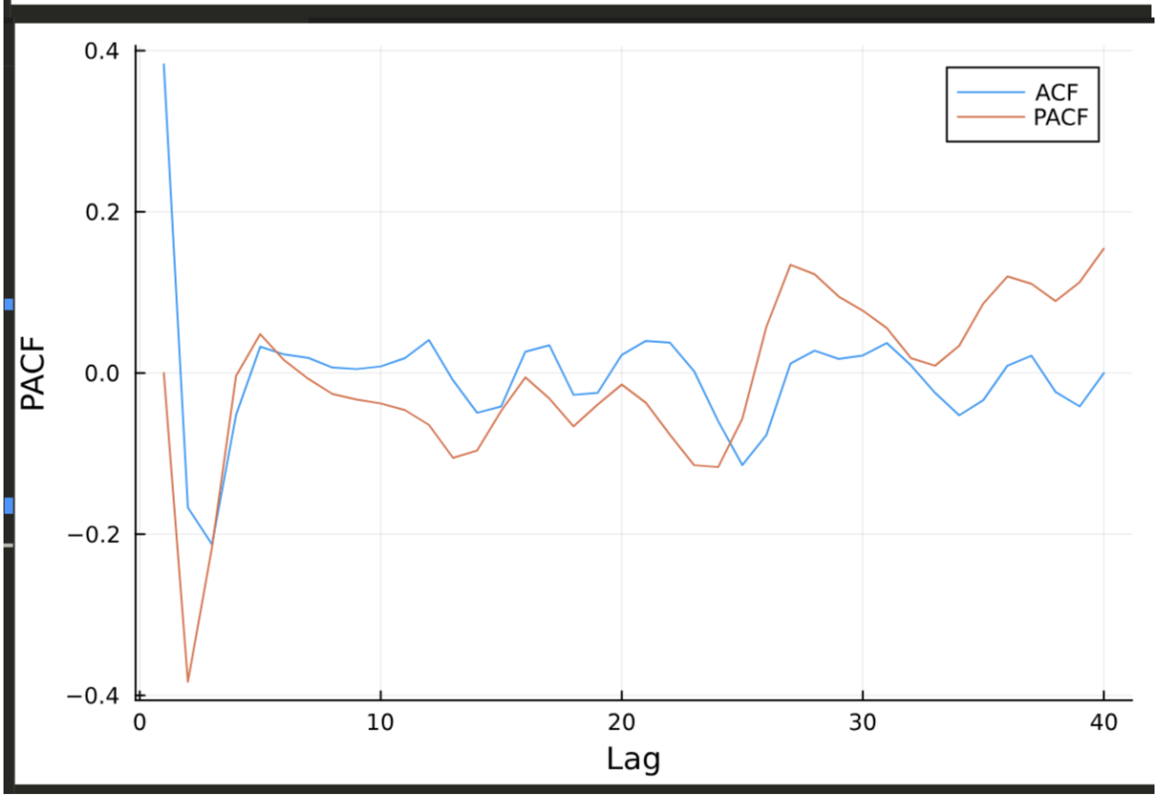




For AR2:

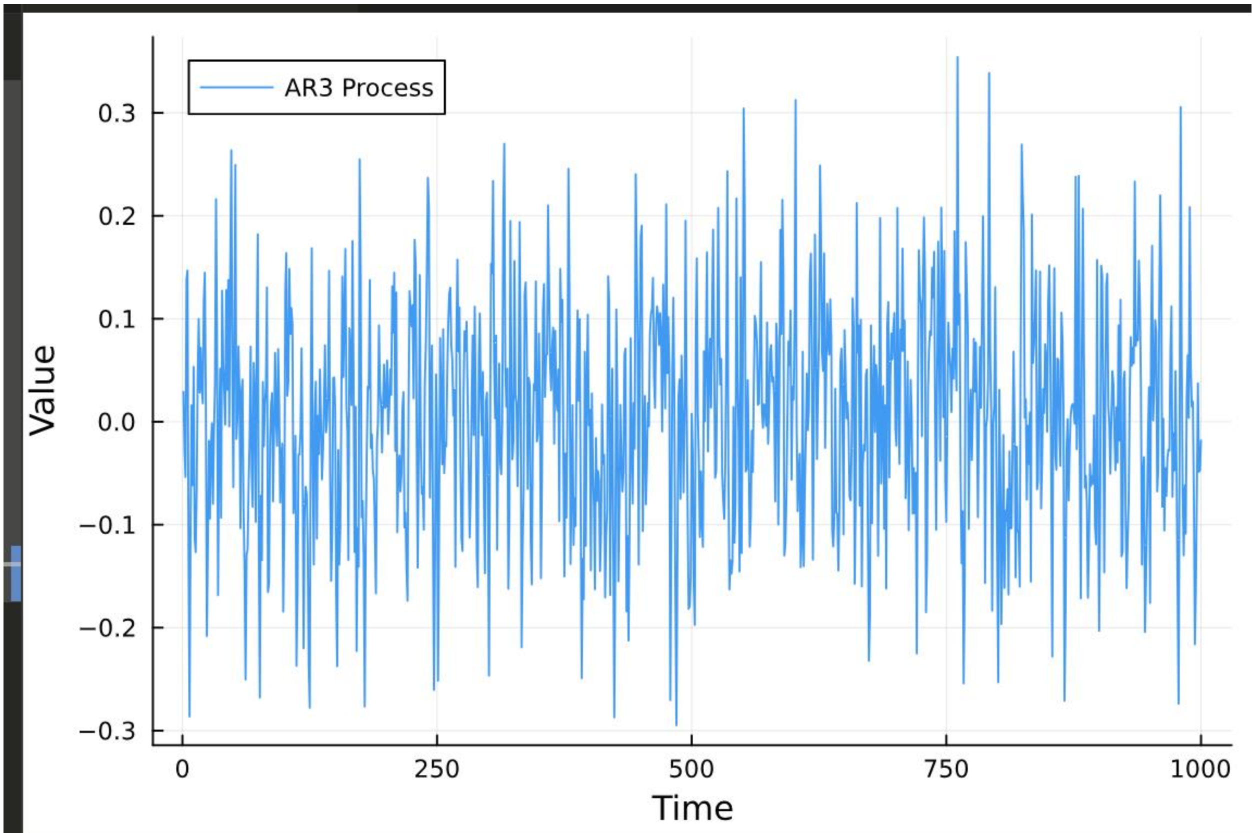


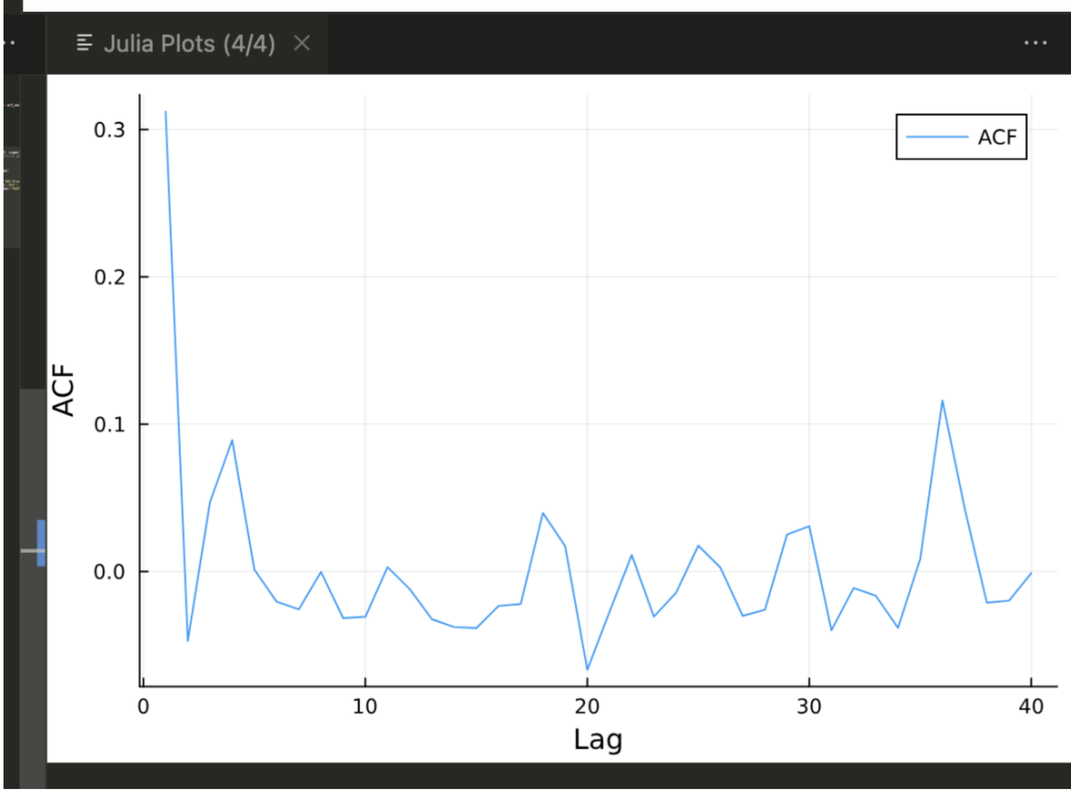
ACF
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
0
10
20
30
40
Lag

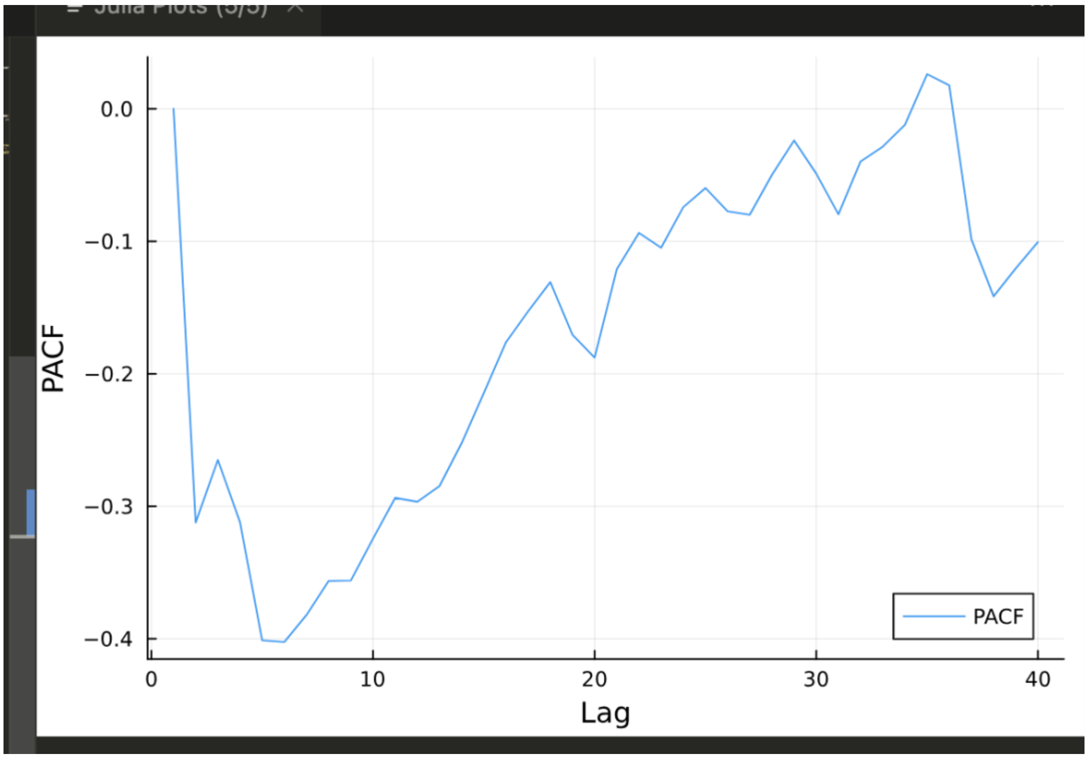


ACF
PACF
0.4
0.2
0.0
−0.2
−0.4
0
10
20
30
40
Lag

For AR3:

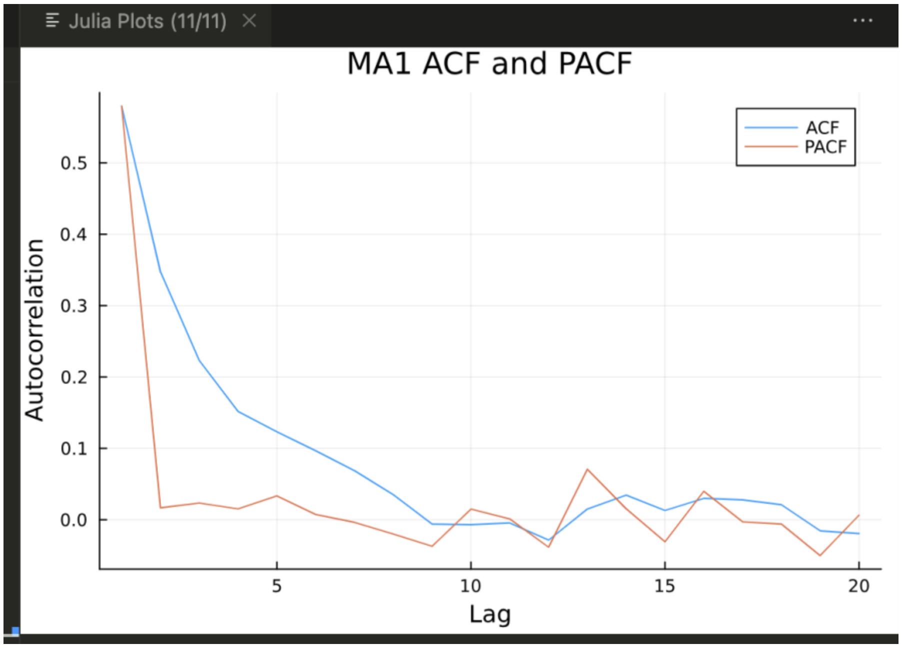

AR3 Process
Value
Time
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
250
500
750
1000


Julia Plots (4/4)
ACF
ACF
Lag
0.3
0.2
0.1
0.0
0
10
20
30
40

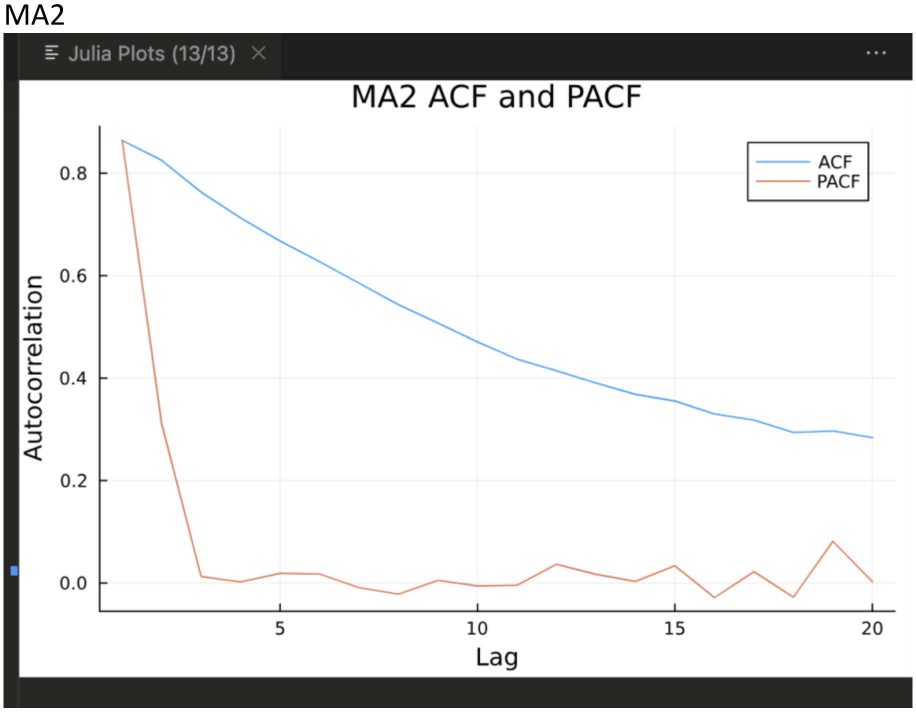


From the above graphs from AR1, AR2, and AR3, we could find that ACF has an exponentially decreasing trend as N in AR(N) increase.
In PACF, significant partial autocorrelations at specific lags help confirm the order of the autoregressive process: a significant partial autocorrelation at lag 1 indicates an AR1 process, at lags 1 and 2 suggest an AR2 process, which could be seen on the above graohs.

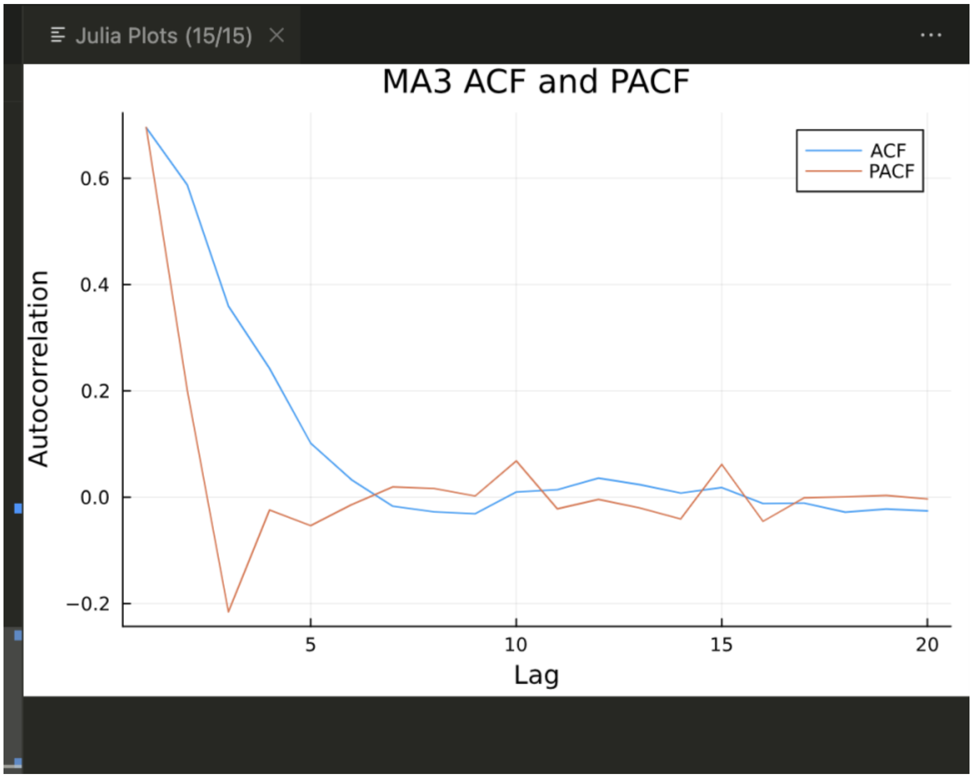
MA1


Julia Plots (11/11)
MA1 ACF and PACF
ACF
PACF
Autocorrelation
0.5
0.4
0.3
0.2
0.1
0.0
5
10
15
20
Lag

MA2


Julia Plots (13/13)
MA2 ACF and PACF
ACF
PACF
Autocorrelation
0.8
0.6
0.4
0.2
0.0
5
10
15
20
Lag

MA3



Through analyzing the decay trend in the ACF and PACF plots, we could detect the type and order of MA1, 2, and 3. According graphs, we could see a perfect correlation for lag 0. And the curves on the graphs are gradually flats and close to 0 as N for MA(N) gets larger.