Time series analysis on US interest rates

Patrik Franzén Dennis

January 2024

Contents

1 Introduction

Time-dependent stochastic processes are fundamental in describing a large variety of natural phenomena. Finance is a field where such processes are used to describe the price of a stock, derivative or interest rates. In this study, tools provided by the theory of stochastic processes are used to used to describe and forecast US interest rates between July 1983 and August 2008.

2 Background

A stochastic differential equation (SDE) can be used to describe interest rates. Such an SDE is the statistical model named, The Vasicek model,

$$dr_t = a(br_t)dt + \sigma dW_t.$$

The above equation denotes the interest rates as r_t, with the initial value r_0. Furthermore, $ab - r_t$, denotes the expected change in the interest rate, and σ denoting the volatility of the changes, i.e., influencing the randomness of r_t. Additionally, W_t denotes a Brownian motion describing the random market risk. An important fundamental property of the model, is the model being equipped with mean inversion. Mean inversion means that the interest rate will move back to the long-term mean b. The reversion speed is governed by the coefficient a.

A discrete version of the Vasicek model can be written as,

$$R_t = ab + \phi R_{t-1} + Z_t,$$

where Z_t is normally distributed IID noise with $Z(0, \sigma^2)$. By starting at the above discrete model, one can see that the presence of an autoregressive process of order 1.

3 Data preprocessing and other calculations

We want to first load the data and calculate its sample mean $\hat{\mu}$ and the mean-corrected time series $S_t \coloneqq R_t - \hat{\mu}$, implemented below.

```
interest <- read.csv('interest.csv')
interest.vec <- c(interest[,1])
sample.mean <- mean(interest.vec)
```

The above code first reads the csv-file containing the interest rates for the study. The csv is then stored as dataframe named *interest*. In order to apply numerical functions and methods on

the provided data, one must first vectorize the values in the dataframe. The changed datastructure is thereafter stored as the variable *interest.vec*. Now the sample mean $\hat{\mu}$ can be computed by using the built-in R function *mean*, resulting in $\hat{\mu} = 5.444073$.

After computing the sample mean, one is to define a new set of values, being the mean-corrected time series defined by,

$$S_t \coloneqq R_t - \hat{\mu}.$$

To obtain the defined S_t one can provide the following code.

```
S_t <- interest.vec - sample.mean
```

The above uses the fact that the interest rates are already stored as a vector and R's capability of detecting when element-wise operations should be performed. With the code above, each element is subtracted with the sample mean computed, thus providing S_t.

3.1 The ACF & PACF

Now that S_t is obtained, one can now find and plot the autocorrelation- and partial-autocorrelation functions. One way to go about such a task is to make use of functions provided by the forecast library. The forecast library has two functions which are useful for plotting both of the desired functions; ggAcf and ggPacf. These functions provided by the forecast library simplifies the process of acf- and pacf plotting, where two parameters are to be provided: Time-series values and the maximum amount of lag. As seen in the code below the two parameters are S_t for the time-series and 20 for the maximum lag. Using the above functions one can produce the ACF- and PACF plots with the code segment below.

```
plot <- ggAcf(S_t, lag.max = 20)
print(plot)

plot <- ggPacf(S_t, lag.max = 20)
print(plot)
```

Note The functions *acf* and *pacf* can also be used to provide the plots of the functions. GgAct and GgPacf are used for visualisation preferences of the writer. Note that the lag starts at $t = 1$. This is due to $t = 0$ always having the value 1. We therefore omit this and focus our analysis on all values for $t \geq 1$.

The code above provides one with the following results.

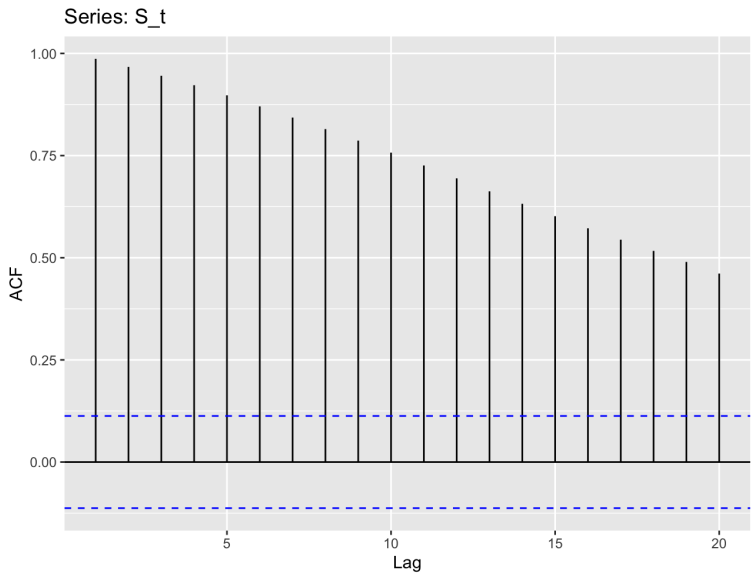

Figure 1: The figure shows the autocorrelation function for the time-series S_t. The dashed blue lines provide the values beyond which the autocorrelations are 95 % (statistically) significantly different from zero, with the confidence interval $[-1.96/\sqrt{n}, 1.96/\sqrt{n}]$

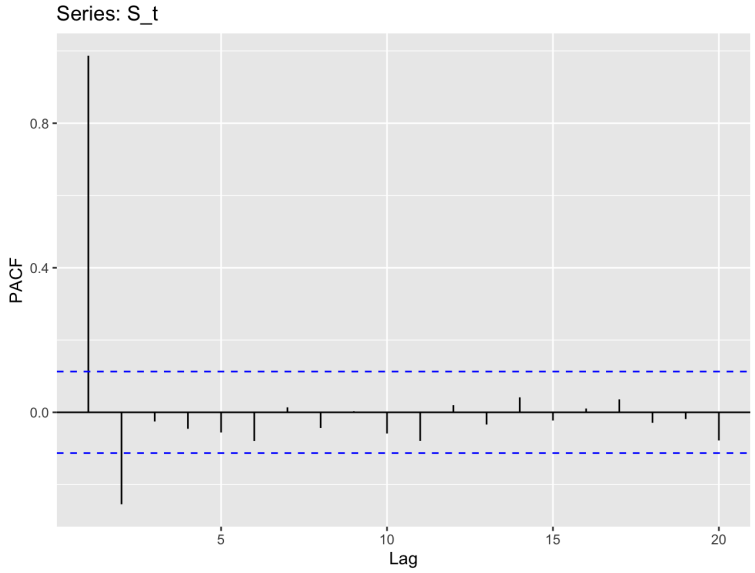

Figure 2: The figure shows the partial-autocorrelation function for the time-series S_t.

From Figure [1], one can see that the ACF is decreasing for every consecutive lag. Figure [1] also shows how the next time point is dependent on the previous time point. Such a decreasing

nature of the ACF gives reason to believe that the time-series is auto-regressive of a certain order p. By observing the latter figure [2] showing the PACF, one can see how the the lags $h \in [0, 1]$, i.e., 0 and 1, are significantly different from 0. After lag 2 there is a sharp decrease rendering the significance negligible which proceeds to be the case for all remaining lags. Due to the PACF being significantly different to 0 for all lags $0 \leq n \leq 2$, provides cause to reason that the auto-regressive model is AR(2).

4 Assuming an AR(1) process

Assuming that the time series model is given by $S_t = \phi S_{t-1} + Z_t$ should hold for some parameters ϕ and σ. This is a linear model, where S_t is called the dependent variable and S_{t-1} the independent variable in this context. The goal of this task is to estimate ϕ and σ with linear regression.

4.1 Investigating linearity

Since our model considers the pairs $((S_{i-1}, S_i))_{i=2}^N$, we want to investigate such pairs and see if a linear relationship seems reasonable.

The first task is to create a data set containing the pairs $((S_{i-1}, S_i))_{i=2}^N$. To obtain such a data set one can use the function *lag*. Lag is a function that takes in two parameters: data-frame and lag. The first parameter is the data frame that contains the values one wants to obtain the lag for, and the second parameter being the desired lag-size. In this study and task, the lag is of size 1. Before using the lag function one must first create a data-frame with the values of interest, where the values of interest are the S_t vector created previously. From the code segment below one can see how the S_t vector is stored into a data frame *S_t.df*. Note that the time points are included in the data frame to validate that the lags have been computed correctly.

Applying the lag function results in a shifted data frame by the provided amount of lag one specified, i.e., 1. The shifting of *S_t.df* by 1 is stored as *lag.interest*. In order to have clear and readable names for the data set, renaming of *S_t* to *S_(t-1)* and time to time lag was performed. It is now clear what columns of the data set represent, thus mitigating potential readability and clarity issues. Now with the time shifted values of S_t stored in the column *S_(t-1)*, one can add these values to the original data frame. Finally, another data set is defined, which holds the same values as the original data set *S_t.df*, but excluding the first row. The reason for this is the first row having a missing value, due to the time shifted values not having anything to shift.

```
#CREATING DATA-FRAME TO GET LAG
time <- 1:302

S_t.df <- data.frame(time, S_t)

lag.interest <- lag(S_t.df, 1)
lag.interest <- lag.interest %>% rename('S_(t-1)' = 'S_t', 'time lag' = 'time')
S_t.df['S_(t-1)'] <- lag.interest$`S_(t-1)`
S_t.df2 <- S_t.df[2:nrow(S_t.df), ]

```

Now that the appropriate data sets have been created, they can be used to plot the desired scatterplot with the pairs $((S_{i-1}, S_i))_{i=2}^{N}$. The code segment below provides a method to perform such a task, resulting in the figure below.

```
#PLOTTING SCATTERPLOT TO SEE CORRELATION BETWEEN VALUES
ggplot(S_t.df[2:nrow(S_t.df),], aes(x=`S_(t-1)`, y= `S_t`)) +
  geom_point(color = 'blue')
```

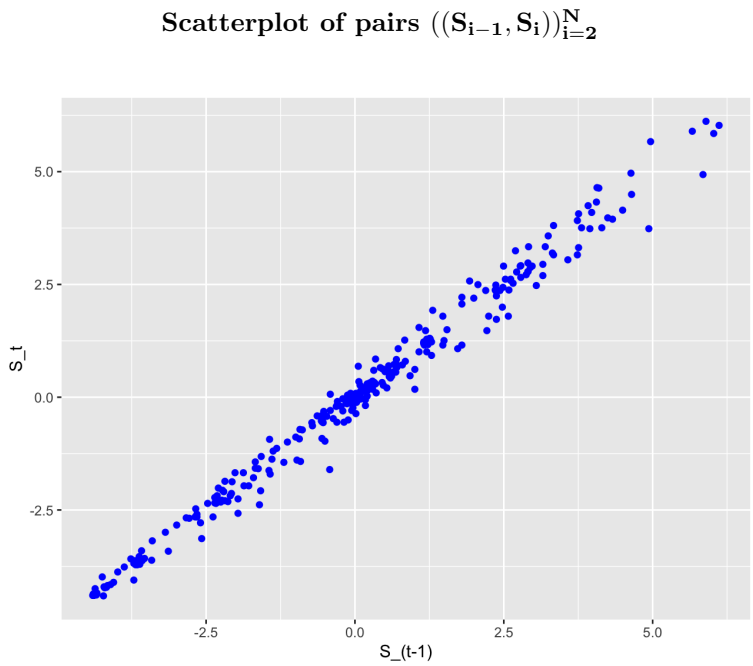

Figure 3

Figure [3] shows a correlation between the pairs $((S_{i-1}, S_i))_{i=2}^{N}$. This being due to the linear relationship of the points, thus showing a trend regarding the pairs. Therefore, one can conclude that the points are correlated to one another.

5 Parameter estimation and sample paths

Using our data set, we want to estimate ϕ and σ using linear regression without intercept. Thereafter we want to simulate and plot a few sample paths of the estimated model together with the mean-corrected data, to fully get a gist of our forecast.

5.1 Estimating ϕ

The first task is to use the obtained data set *S_t.df*, to estimate ϕ and σ. The method used to estimate the two parameters is by linear regression. To obtain estimates of the two parameters by linear regression, one can use the *lm* function in R. Due to the original data set S_t.df possessing a missing value in the first row, the data set S_t.df2 can be used. The task also states that no intercept should be included. One can exclude the intercept by adding a zero as the first term in the model provided in the lm-function. The performed linear regression to obtain $\hat{\phi}$ is stated below.

Linear Regression to Obtain $\hat{\phi}$

```
1 #PERFORMING LINEAR REGRESSION TO GET PSI
2 l_reg <- lm(S_t.df2$S_t ~ 0 + S_t.df2$`S_(t-1)`)
3 phi <- l_reg$coefficients
4 phi <- 0.9908014
```

After running and storing the results provided by the regression model in *l_reg*, one can retrieve the estimated ϕ by obtaining the resulting coefficients. This achieved by running line 3 of **??**. The estimated ϕ obtained by the model above is, 0.9908014.

5.2 Estimating σ

Now that $\hat{\phi}$ is known, one can obtain the estimate of σ. Due to,

$$Z_t = S_t - \phi S_{t-1}. \tag{1}$$

Therefore, estimation of σ is computed by,

$$Var(S_t - \hat{\phi} S_{t-1}). \tag{2}$$

The variance above is computed in R show below, resulting in $\hat{sigma} = 0.2706224$.

```
1 #OBTAINING SIGMA
2 Z.sigma2 <- var(S_t.df2$S_t - psi*S_t.df2$`S_(t-1)`)
3 Z.sigma <- sqrt(Z.sigma2)
```

5.3 Simulating Sample paths

Given the estimated values, $\hat{\phi}$ and $\hat{\sigma}$, sample paths can now be simulated. Sample paths are eventual paths the time-series can take for future values. Sample paths can here be performed

due to the recursive nature of $S_t = \phi S_{t-1} + Z_t$. In this study the future values represent time-points further than time point 302.

The code below is divided into three sections, with each representing individual sample paths. The method is analogous for all three sample paths. The sole discrepancy of the three sections is the initial creation of the data frame, which will store all three sample paths. This is seen in the first code segment where the R function *tibble* is used to create a data frame using the library dplyr.

Sample path 1

```
S_t.samplepath1 <- S_t.df2$S_t

for (i in seq(0,50,1)){
  S_t.samplepath1[302+i] <- psi*S_t.samplepath1[(301+i)] + rnorm(1,mean=0,sd=Z.sigma)
}

df.samplepath <- tibble(S_t.samplepath1)

```

Sample path 2

```
S_t.samplepath2 <- S_t.df2$S_t

for (i in seq(0,50,1)){
  S_t.samplepath2[302+i] <- psi*S_t.samplepath2[(301+i)] + rnorm(1,mean=0,sd=Z.sigma)
}

df.samplepath['samplepath2'] <- S_t.samplepath2
```

Sample path 3

```
S_t.samplepath3 <- S_t.df2$S_t

for (i in seq(0,50,1)){
  S_t.samplepath3[302+i] <- psi*S_t.samplepath3[(301+i)] + rnorm(1,mean=0,sd=Z.sigma)
}

df.samplepath['samplepath3'] <- S_t.samplepath3
```

Sample Path 4

```
S_t.samplepath4 <- S_t.df2$S_t

for (i in seq(0,50,1)){
  S_t.samplepath4[302+i] <- phi*S_t.samplepath4[(301+i)] + rnorm(1,mean=0,sd=Z.sigma)
}

df.samplepath['samplepath4'] <- S_t.samplepath4
```

The methodology used to create the sample paths, is to iterate with a step size of 1, where the iterated values are the previous values computed, i.e., by recursion. Further, whilst iterating, a random number is provided by the function *rnorm*, which provides a random number from the standard normal distribution, where one can specify the mean and standard deviation. For this task and study the mean is 0 and the standard deviation is the estimated $\hat{\sigma}$. This process is performed three times, thus resulting in three different sample paths. Note that the path size is 50.

Sample Path Plot

Figure 4: The figure shows three different sample paths. The first simulated sample paths past the time point 302.

We can see how each forecast varies wildly between each other, illustrating the risk in the dark arts we call forecasting.

Note Observe that the green line prior to the prior to the 303 time point is the original time series and is not a sample path. The green line after the 302 time point is however a sample path.

6 Residual analysis

Given the our parameter estimates, it is high time to perform residual analysis. Thus the first task is to calculate and plot the residuals, $E_i := S_{i+1} - \hat{\phi} S_i$. One can observe that the methodology of calculating the residuals is analogous to the method for retrieving $\hat{\sigma}^2$.

```
#FINDING THE RESIDUALS
residuals <- S_t.df2$S_t - psi*S_t.df2$`S_(t-1)`
df.residuals <- tibble(residuals)
```

The code above finds the residuals by subtracting the S_t-values with the the estimated ϕ multiplied with S_{t-1}. After computing the respective residuals, one can now plot the residuals.

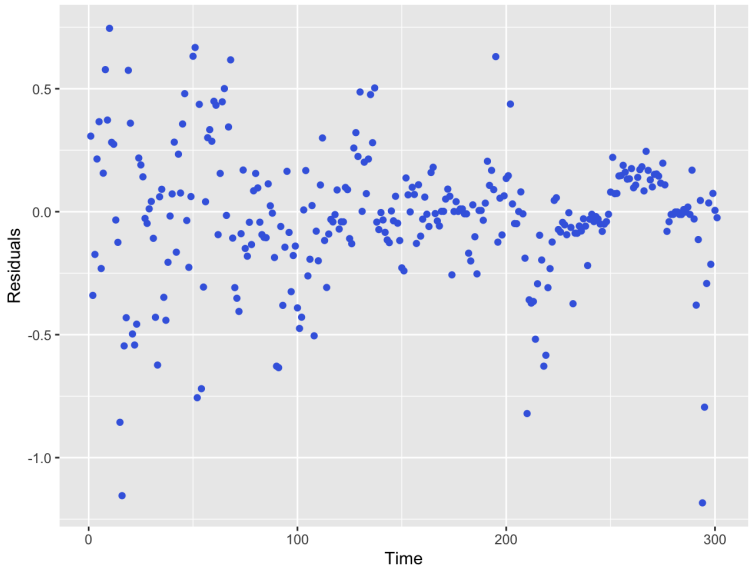

Figure 5: Displaying the residual plot for E_i.

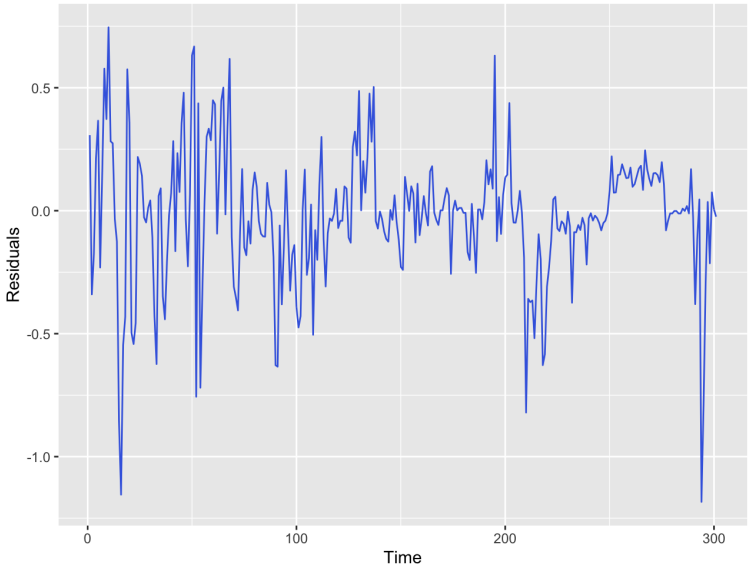

Figure 6: Continuous plot of residuals

From the above plots, it seems reasonable to assume that the residuals E_i are iid. This being due to Figure [5] having no apparent linearity or correlation. To further investigate iid, one can perform the Ljung-Box Test, shown below.

```
1 Box.test(residuals, type="Ljung",lag=25)
```

Data:	Residuals
χ^2	118.66
df	25
p-value	3.819e-14

Table 1: Results obtained by the Ljung-Box test statistic.

From the results provided above, if one were go purely on the Ljung-Box test, it suggests a significant rejection of independence. This also seems to be the case if one plots the autocorrelation function.

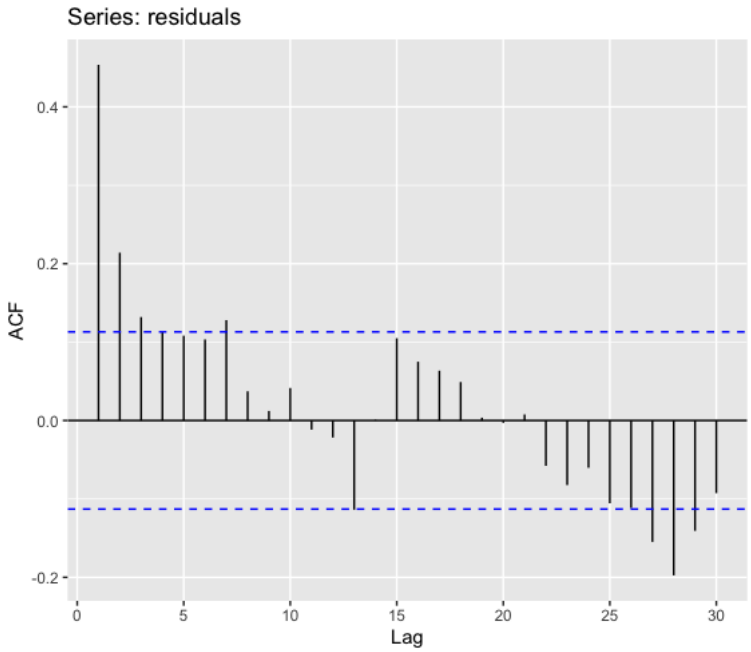

Figure 7: Acf for the residuals with maximum lags of 30

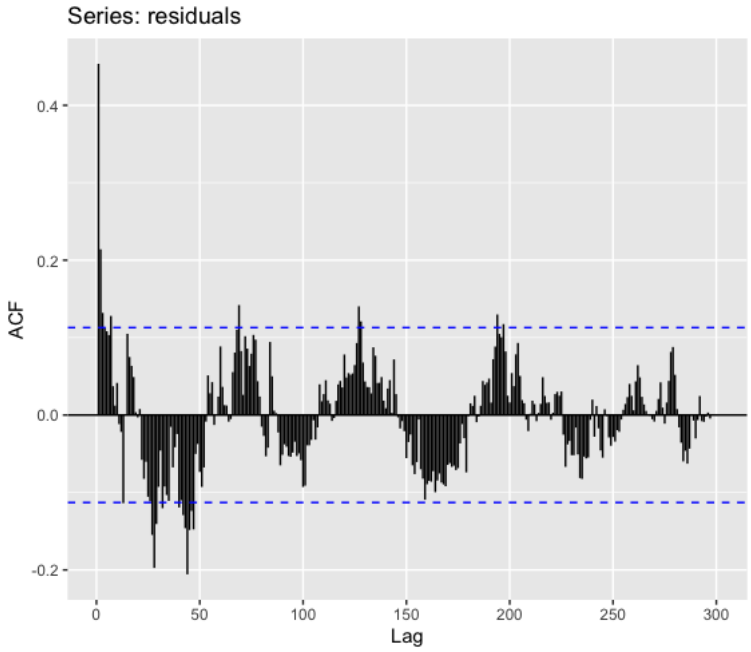

Figure 8: Acf for the residuals with maximum lags of 301

The figure 7 displays 84 % lying within the 95% interval. The above also suggests not being

iid. Figure reference suggests periodicity in regards to the residuals. When the noise is not iid, as this case suggests, the appropriate step to take is model reevaluation. Furthermore, a possible route to take would be to difference the data until iid data appears. This means that one uses differencing to eliminate any potential seasonality. Such an approach is shown in subsequent section.

6.0.1 Eliminating Potential Seasonality

As mentioned, we can difference our data until we obtain iids data. An implementation of this is seen below resulting in figures [9] and [10].

```
df <- diff(S_t.df2$S_t, lag=1, differences=1)
s_t_1 <- diff(S_t.df2$`S_(t-1)`, lag=1, differences=1)

l <- lm(df ~ 0 + s_t_1)
res <- residuals(l)
ggAcf(res)

```

From figures [9] and [10], one can see how the structure of the residuals has changed. This is especially seen in figure (10) showing the ACF of the residuals from the now differenced series. Now the percentage of values exceeding the boundaries is at 8%. This further suggests that there may well be a seasonality component to the interest rates. This is a widely discussed subject amongst researchers in economics, where some believe in seasonality regarding interest rates, and others criticize this.

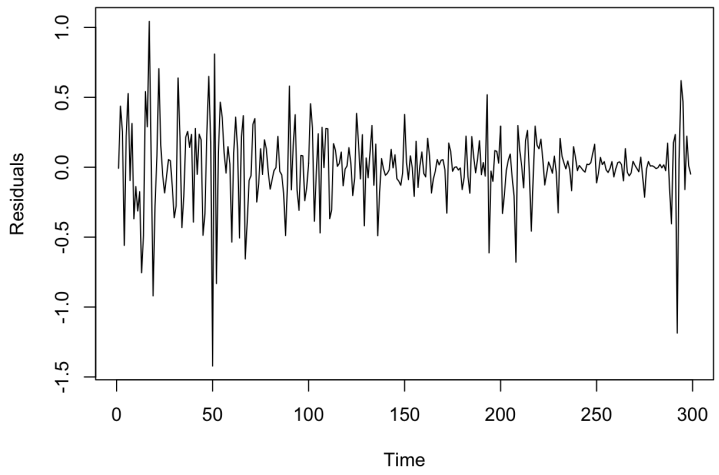

Figure 9: Plot showing residuals of (Non-seasonal) differenced residuals and series.

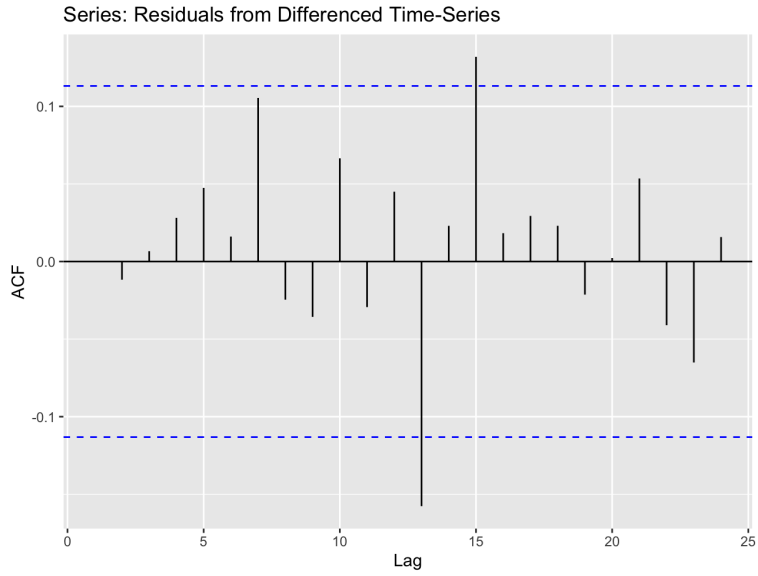

Figure 10: Plot showing ACF of differenced series of residuals.

Now computing the Ljung-Box test for the differenced data provides a p-value of, 0.9979. Therefore, with this one fails to reject the null hypothesis of being IID.

7 Forecasting & The Best Linear Predictor

The main objective for this task is to forecast a future value by taking the previous 20 values into consideration, i.e., predicting a value one month in advance with the knowledge of the previous 20 months. In order to accomplish such a forecast one must use predictors, and thereafter evaluate such predictors. To test the eventual predictors splitting the data into a training and test set is essential. In this task a train-test-split is performed on the S_t data set, where the first 201 data points are stored as training data and the remaining data points as test data. Such a split is performed as seen in the code segment below.

```
training.set <- S_t[1:201]
test.set <- S_t[202:302]
```

The next step is deciding the best linear forecast for a future month given the last 20 months,

$$b_n^l(z_{n-1}, ..., z_{n-20}) \coloneqq a_1 z_{n-1} + a_2 z_{n-2} + ... + a_{20} z_{n-20}. \tag{3}$$

In order to achieve finding the best linear predictor for the above, the following proposition below is used to find the unknown a_n coefficients, where the z_n are the known data points.

Proposition 2.4.5 Let $(X_t, t \in \mathbb{Z})$ be a time series with $Var(X_t) < \infty$ for all $t \in \mathbb{Z}$ and $X_n \coloneqq (X_{t1}, ..., X_{tn})$ a collection of random variables of the time series at n different times. Then the best linear predictor of X_t is given by $b_l^t(X_n) = a_0 + a_1 X_{tn} + a_2 X_{tn1} + + a_n X_{t1}$, where the coefficients $(a_i, i = 0, ..., n)$ are determined by the linear equations,

- $\mathbb{E}(X_t b_l^t(X_n)) = 0$,
- $\mathbb{E}(X_{t_j}(X_t b_l^t(X_n))) = 0$ for all $j = 1, ..., n$.

Corollary 2.4.6, provided below, simplifies the set of equations defined above, to be solved to one equation.

Corollary 2.4.6 Let $X = (X_t, t \in \mathbb{Z})$ and X_n be as in Proposition 2.4.5 and assume in addition that X is stationary with mean μ and autocovariance function γ. Then the coefficients $(a_i, i = 0, ..., n) of b_l^t(X_n)$ are determined by the linear equations,

$$a_0 = \mu\left(1 - \sum_{i=1}^{n} a_i\right) \tag{4}$$

and,

$$\Gamma_n(a_1, ..., a_n)' = (\gamma(t - t_n), ..., \gamma(t - t_1))', \tag{5}$$

with $\Gamma_n = \gamma(t_{n+1-j} - t_{n+1-i}))_{i,j=1}^n$.
With Corollary 2.4.6, one can use (4) and (5) to find the desired coefficients a_n.

Note The best linear predictor that is to be found, i.e., 3, does not include a_0. This being due to μ being 0, thus resulting in a_0 also being equal to 0.

Toeplitz Matrix In order to obtain Γ_n, the definition of a Toeplitz matrix is used. The reason for this is due to autocovariance matrix being a Toeplitz matrix, which enables the use of the *toeplitz* function in R to construct Γ_n.

Definition Toeplitz Matrix A Toeplitz matrix is a matrix in which each descending diagonal is constant. The Toeplitz matrix has the following form,

$$\begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & a_{-(n-1)} & \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \cdots & \cdots & a_2 & a_1 & a_0 \end{bmatrix} \tag{6}$$

With the structures of (7) and (6) in mind, one can conclude that Γ_n is indeed Toeplitz.

With the information provided it is now possible to implement (5) in order to find the coefficients a_n for the best linear predictor.

```
1 acvf <- acf(training.set, type = "covariance", lag.max = 20)
2 acvf.matrix <- toeplitz(c(acvf$acf[1:20]))
3
4 coeff <- solve(acvf.matrix,acvf$acf[2:21])
```

From the above code the sample autocovariance values are computed to be used for constructing Γ_n. Thereafter, the solution is computed on the fourth line. The solution provided by the above is,

$$
\begin{aligned}
a_1 &= 1.279010795 & a_{11} &= -0.178761545 \\
a_2 &= -0.345993915 & a_{12} &= 0.096611713 \\
a_3 &= 0.039622130 & a_{13} &= -0.168086521 \\
a_4 &= 0.032396664 & a_{14} &= 0.159462415 \\
a_5 &= 0.039712601 & a_{15} &= -0.034809088 \\
a_6 &= -0.077320215 & a_{16} &= -0.061301176 \\
a_7 &= 0.059965570 & a_{17} &= 0.104650223 \\
a_8 &= 0.003116343 & a_{18} &= -0.068129366 \\
a_9 &= -0.102814113 & a_{19} &= 0.156612376 \\
a_{10} &= 0.175165305 & a_{20} &= -0.144978274
\end{aligned}
$$

First it should be noted how the predictor is defined, where the first coefficient is to be multiplied by the last value in the collection of 20 data points provided as information to predict upon. With this in mind one can therefore rewrite $a_1 z_{n-1} + a_2 z_{n-2} + ... + a_{20} z_{n-20}$ as,

$$
b_n^l(z_n) = \begin{bmatrix} z_{n-1} & z_{n-2} & \cdots & z_{n-20} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{20} \end{bmatrix} \tag{7}
$$

Using (7), one can recursively predict one month at a time with the previous 20 values. Meaning that once z_n is predicted, z_{n+1} can be predicted with now $(z_n, z_{n-1}, ..., z_{n-19}$ and so on and so forth. To implement the reasoning carried out previously, the below R-code is used.

R-Code Best Linear Prediction Implementation

```
for (i in 202:302){
  training.set.sample[i] <- coeff %*% rev(c(S_t.df$S_t[(i-20):(i-1)]))
}
```

From the above one can see how each new value to be predicted in the training set uses the formula [3]. Note that the function *rev* is used, which reverses a list or vector, in this case being the data points. The reason for reversing the array is due to the definition of the linear predictor as noted previously.

Performance evaluation of $b_n^l(z_n)$

Now that the predicted values have been computed with the best linear predictor, evaluation of the predictors performance is due. The first step is to plot the results, and compare the nature of the values in comparison to the actual data points. To reader is reminded that the actual or true data points are stored in the test set. Running the code below leads to the figure (reference).

```
sample.bestpred <- training.set[202:302]

to.test <- tibble("Sample best predictor" = sample.bestpred, "Test"=test.set)

plot <- ggplot(to.test, aes(x=1:101)) +
  geom_line(aes(y = `Sample best predictor`, colour='Sample best predictor')) +
  geom_line(aes(y=Test, colour = "Test")) +
  geom_line(aes(y=0, colour = "mean")) +
  scale_colour_manual("",
                      breaks = c('Sample best predictor',"Test","mean"),
                      values = c('Sample best predictor'="royalblue", "Test"="orange1",

                      "mean"="green")) +
  labs(x="Time", y = TeX("$S_t$"))
print(plot)
```

Test vs Predicted

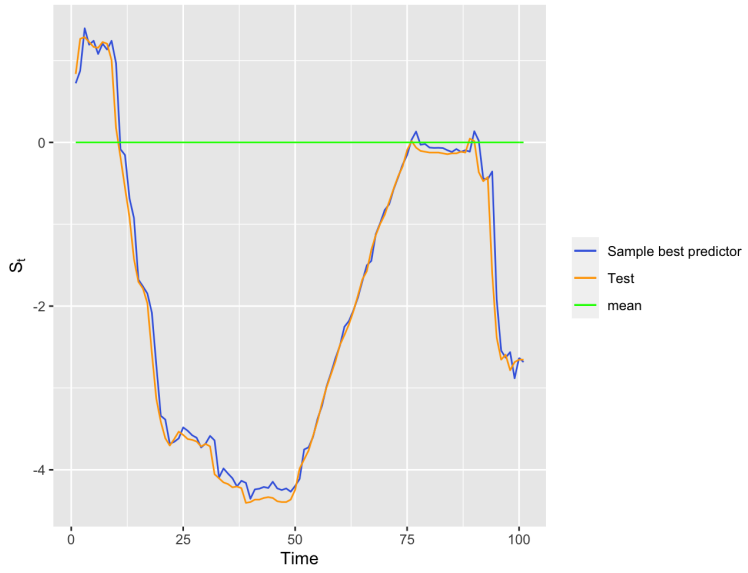

Figure 11: Plot showing the predictions (blue) obtained via $b_n^l(z_n$ vs the test data (orange) and mean (green).

Figure [11] shows that the linear predictor follows the motions of the test set well, meaning the series representing the test data is in close proximity to the predicted values via the best predictor.

Error measurements

To not only base conclusions on mere eyeballing, one can use error metrics to evaluate the performance. The chosen metric is the mean squared error. A generic function was used to ease the process in computing multiple mean squared errors.

Function to Compute Mean Squared Error

```
mse <- function(real, predicted){

  if(length(predicted)==1){
    predicted <- integer(length(real))
  }
```

```
  s <- 0
  difference <- NULL
  for (i in 1:length(real)){
    difference <- (real[i] - predicted[i])^2
    s <- s + difference
  }
  return(s/(length(real)+1))

}
```

The code [7] uses the definition of mean squared error. The definition of the mean squared error for the current task is,

$$\frac{1}{101}\sum_{n=202}^{302}\left(b_l^n(z_{n-1},...,z_{n-20})-z_n\right)^2. \tag{8}$$

Applying the function *mse* to the test set and the predicted values, results in the error being, 0.04503199. This is a low error when regarding to the range of values for the data. To gauge whether the linear forecast has performed well, a comparison to a more naïve forecast is performed. The chosen naïve forecast is forecast via the mean. Since the data used is mean corrected, the mean will result in being 0. Due to the mean being 0, the mean squared error for such a model equates to,

$$\frac{1}{101}\sum_{n=202}^{302}\left(\hat{\mu}-z_n\right)^2=\frac{1}{101}\sum_{n=202}^{302}\left(z_n\right)^2. \tag{9}$$

Using the above to compute the error results in, 7.254891. When comparing the two, the naïve performs worse in regards to the error metric. If one was to purely base results on the error metric, the linear predictor prevails.

8 I predict more predictions

We will now make the same predictions as in the previous section, but we use the parametric assumption that S follows an autoregressive process of order 1 with the parameters $\hat{\phi}$ and $\hat{\sigma}$ that were estimated. We want to calculate the best linear predictor in this case. This meaning, instead of using the sample ACVF $\hat{\gamma}$, we want to take the ACVF with the estimated parameters $\hat{\phi}$ and $\hat{\sigma}$. Thereafter we want to plot the predictions in the same figure as the test data and calculate the same errors as in the previous exercise. Have we improved the quality of the predictions?

The task above is now to use the theoretical autocovariance-function, instead of the sample autocovariance-function to compute obtain the best linear predictor. Therefore, the first step in finding the best linear predictor is to find the theoretical ACVF. Due to the parametric assumption that S_t follows an AR(1)-process, the known ACVF for an AR(1) can be used,

ACVF of an AR(1)-Process

$$\begin{cases} \gamma_{S_t}(0) = \frac{\sigma^2}{1-\phi_1^2} \\ \gamma_{S_t}(h) = \frac{\sigma^2 \phi^{|h|}}{1-\phi_1^2} \end{cases}. \tag{10}$$

The above and using the values: $\phi = 0.9908014$ and 0.2706224, thus renders the equation given by corollary 2.4.6 to,

$$\begin{bmatrix} \gamma(0) & \gamma(1) & \cdots & \gamma(19) \\ \gamma(1) & \gamma(0) & \ddots & \gamma(18) \\ \vdots & \ddots & \ddots & \gamma(17) \\ \gamma(19) & \gamma(18) & \cdots & \gamma(0) \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{20} \end{bmatrix} = \begin{bmatrix} \gamma_{S_t}(1) \\ \gamma_{S_t}(2) \\ \vdots \\ \gamma_{S_t}(20) \end{bmatrix} \tag{11}$$

Resulting in the below,

$$\begin{bmatrix} \frac{\sigma^2}{1-\phi_1^2} & \frac{\sigma^2\phi^{|1|}}{1-\phi_1^2} & \cdots & \frac{\sigma^2\phi^{|19|}}{1-\phi_1^2} \\ \frac{\sigma^2\phi^{|1|}}{1-\phi_1^2} & \frac{\sigma^2}{1-\phi_1^2} & \ddots & \frac{\sigma^2\phi^{|18|}}{1-\phi_1^2} \\ \vdots & \ddots & \ddots & \frac{\sigma^2\phi^{|17|}}{1-\phi_1^2} \\ \frac{\sigma^2\phi^{|19|}}{1-\phi_1^2} & \frac{\sigma^2\phi^{|18|}}{1-\phi_1^2} & \cdots & \frac{\sigma^2}{1-\phi_1^2} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{20} \end{bmatrix} = \begin{bmatrix} \frac{\sigma^2\phi^{|1|}}{1-\phi_1^2} \\ \frac{\sigma^2\phi^{|2|}}{1-\phi_1^2} \\ \vdots \\ \frac{\sigma^2\phi^{|20|}}{1-\phi_1^2} \end{bmatrix} \tag{12}$$

Now by plugging in ϕ and σ and solving equation 12 , which can be achieved by utilizing a function provided below,

```r
acvf.ar1 <- function(phi,sigma,h){
  ((sigma^2)*phi^(abs(h)))/(1-(phi^(2)))
}

theoretical.acvf <- NULL
theoretical.acvf[1] <- acvf.0

for (i in seq(1,20)){
  theoretical.acvf[i+1] <- acvf.ar1(psi,Z.sigma,i)
}


theoryacvf.matrix <- toeplitz(theoretical.acvf[1:20])
acvf.vec <- theoretical.acvf[2:21]

theory.asol <- solve(theoryacvf.matrix,acvf.vec)
```

Applying the above solution, i.e., theory.asol to compute the best linear forecast, the result is as following,

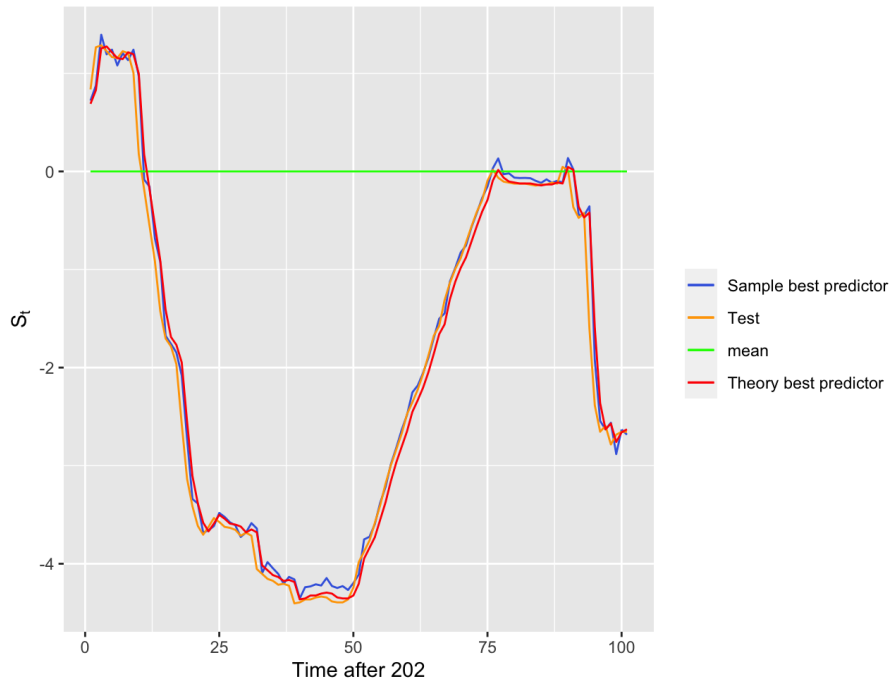

Figure 12: Plot showing the samlple best predictor (blue), test set (orange), naïve prediction (green) and the theoretical best predictor (red).

From figure [12], one can see that the best linear predictor computed with the theoretical acvf and model assumptions provides the same predictions as the best linear predictor computed via the non-parametric route. To further analyze the best linear predictor found in task 5 a summary of all mean squared errors are shown below.

MSE for all predictors

naïve Prediction	Non-parametric b_n^l	Parametric b_n^l
7.254891	0.04503199	0.05753872

Table 2: Table displaying the mean-sqaured-errors for the different predictors.

As seen above, the non-parametric predictor performs superior to the parametric predictor. One should however note that the MSE is a random variable in itself, thus the difference of the error's can be seen to be non-significant, i.e., a result of randomness. Therefore, with the non-significant nature of the errors, one could argue that the error metrics may well be equal. Furthermore, the MSE of the best linear predictor of an AR(1) process is constant. This is due to the mean squared error for a One-Step prediction of an AR(1) series being equal to σ^2, which is shown below.

MSE of One-Step Prediction of AR(1) Series

$$\begin{aligned}\mathbb{E}[X_{n+1} - P_n X_{n+1}]^2 &= \gamma(0) - \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \gamma_n(1) \\ &= \frac{\sigma^2}{1-\phi^2} - \phi\gamma(1) \\ &= \sigma^2\end{aligned}$$

The above derivation of the mean squared error confirms that the computed mean squared errors for both best linear predictors are correct. This due to the estimated σ^2 for the provided series $S_t = 0.07323646$, which is approximately equal to the predictors after taking random errors within the errors into consideration. The near equal results regarding the best linear predictors raises the question whether one ought to bother about parametric assumptions.

9 Parametric assumptions. To be or not to be?

When viewing errors provided in [2], the error of the non-parametric assumption comes out on top. A reason for the superiority of the non-parametric predictor can be a result of the restrictions one sets on the data when using assumptions. This being due to the more properties the data should hold, the more restrictions on the data. Therefore, assuming the data follows an AR(1) process, affects the computation of the acf, which is turn affects the best linear predictor. A major issue in forecasting and data analysis is having parametric assumptions for the data. This being due to the risk of attempting to fit the data to the model. Fitting the data to the model violates the fundamental of science where one should never conform ones results to ones thesis. If one was to go blindly in the data, one may find that the data fits better to for example an AR(2). A recommendation would be to not only test and base conclusions on errors, but also on goodness-of-fit tests, such as: R^2-test, AICC-scores and so on and so forth.

Even though parametric assumptions could provide risk in being blind to the truths of the data, parametric assumptions does not have to be pointless. Assuming a certain structural formation, described by a small number of parameters, is computationally faster than performing non-parametric modelling. A drawback to parametric assumptions is the requirement of temporal and spatial covariance structures, which can be cumbersome. This is not the case for for non-parametric models, which only uses the obtained empirical structures. A problem that one must take into consideration when modeling time series data via non-parametric models such as use of regression, neural networks, Long-short term memory models is the structural instabilities of for instance, computed covariance, noise and randomness exists in time series data. The non-parametric methods mentioned could risk prohibit in acquiring the optimal weights or coefficients. An argument for using parametric modeling in forecasting is that one can use for instance an Bayesian approach for the likelihood of parametric modeling failures.

In conclusion parametric assumptions cannot be deemed pointless due to the theory that one gets "for free" when assuming a certain model. Additionally, if faced with larger data and different data, where simple regression techniques may not be as useful, a parametric model is computationally faster than say using neural networks. From the study, a parametric model seems useful to use for evaluating a potential non-parametric model. The use of theoretic properties one gets from parametric assumptions can be used to arrive to conclusions and insights to the time series. An example of this is knowing that the mean squared error for an AR(1) process is σ^2. This fact would not be entirely obvious if one was to go blind into the data.

If one was to observe figure [12], the theoretic best linear predictor seems to follow the actual data better. This further argues the fact that one should not be entirely reliant on error metrics. This meaning that goodness-of-fit and error metrics are not the only factors one must take into consideration when choosing the final model. The most important factor to take into consideration is the how well the model follows the data, and the scalibility of the model chosen.