STAT 447 Project Final Report

Jessie Liang (52819596)

2025-04-08

0. GitHub repo link

https://github.com/jessie-liang/447-project.git

1. Problem formulation

Due to various factors affecting Canadian economic environment (e.g. policy making, international relation, economic development, etc), the consumer price index (CPI) has been climbing yet fluctuating over the past 20 years. CPI is an index describing changes in prices faced by Canadian consumers by comparing the cost of a representative basket of goods and services through time (Statistics Canada, 2019). CPI is widely used to measure living costs of residents and provide insights of inflation. Therefore, it would be beneficial for all economic agents if future CPI index could be predicted accurately so as to warn people to take necessary and timely actions. The problem that this project aims to solve is to build a reasonably accurate prediction model that helps forecast future CPI values, through a Bayesian approach.

CPI data is essentially a time series, where dependence among neighboring observations cannot be ignored. To incorporate this characteristic into our Bayesian framework, state-space model and ARMA model are used for modelling in this project. One of the main challenges is to elaborately capture all features that the time series exhibits, including an upward trend and cyclical variations, and then yield predictions that also show these features. The steps to tackle this problem are: (1) Split the original dataset into a training set and a test set. (2) Decompose the training set using LOESS method (Cleveland et al., 1990) into a trend component, a seasonal component and remainders. (3) Model the trend component using the state-space model and calculate predictions of this component. (4) Model the remainders component using ARMA model and calculate predictions of this component. (5) Obtain point forecasts by summing three parts, namely predictions of the trend component, the corresponding values of the seasonal component, and predictions of the remainders component. (6) Obtain 99% credible intervals by calculating the 0.5% and 99.5% quantiles of the posterior samples of point forecasts. (7) Compare the prediction results with the original test set.

2. Literature review

3. Data analysis

As mentioned above, LOESS decomposition is applied first before we build separate models for each component. The visualization of the decomposition is shown in Figure 1. As we can see, the trend component is smooth and upward sloping, which can be well approximated by a linear regression. The seasonal component is repeating the same pattern again and again, without any randomness involved. Actually the values of the seasonal component are exactly the same in each cycle, so Bayesian modelling is not needed for this deterministic relation. The remainder component seems random with mean zero, and this is where an ARMA model is adopted to model this process.

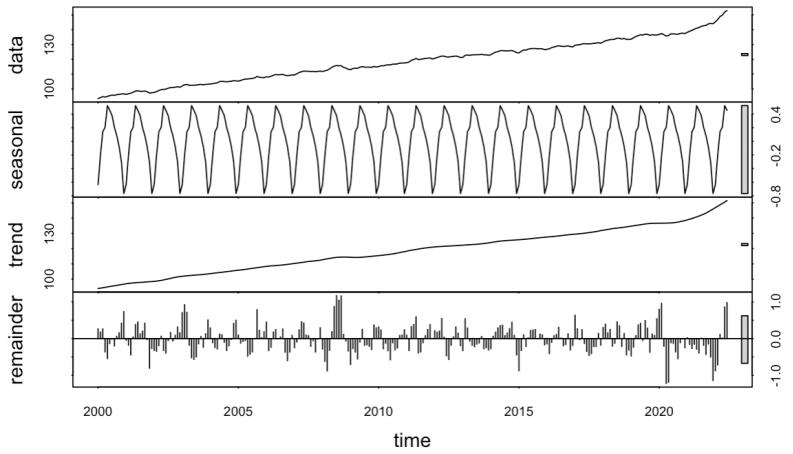


Figure 1: LOESS decomposition

3.1 Trend component modelling

Mathematical model:

$$\sigma_1 \sim \text{Exp}(0.1)$$

$$\sigma_2 \sim \text{Exp}(0.1)$$

$$\text{slope} \sim \text{Normal}(0, 100)$$

$$\text{intercept} \sim \text{Normal}(0, 100)$$

$$x_i \sim \text{Normal}(\text{intercept} + \text{slope} \cdot x_{i-1}, \sigma_1), \text{ for } i = 1, \ldots, N_{\text{train}}$$

$$y_i \sim \text{Normal}(x_i, \sigma_2), \text{ for } i = 1, \ldots, N_{\text{train}}$$

Prior choice explanation: (1) σ_1, σ_2 are positive-valued, which match the support of the exponential distribution. Exp(0.1) is relatively flat, flexibly enabling large values to be considered. (2) slope and intercept are real numbers but are potentially unbounded, so a normal distribution is appropriate here. The standard deviation of the normal is set to 100, making the distribution flat. So large values, either positive or negative, could be taken into consideration.

3.2 Remainder component modelling

An ARMA(2,2) model is adopted here, whose defining equation is: $X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + Z_t + \theta_1 Z_{t-1} + \theta_2 Z_{t-2}$, where $\{Z_t\}$ is a white noise process and $\{X_t\}$ is the process we are interested in. Translating this definition into Stan code, we have the following Stan model:

Mathematical model:

$$\phi_i \sim \text{Normal}(0, 100), \text{ for } i = 1, 2$$

$$\theta_i \sim \text{Normal}(0, 100), \text{ for } i = 1, 2$$

$$\sigma \sim \text{Exp}(0.1)$$

$$\text{err}_0 = 0; \ \text{err}_i = y_i - \nu_i, \text{ for } i = 1, \ldots, T$$

$$\nu_1 = 0, \ \nu_2 = \phi_1 \nu_1 + \theta_1 \text{err}_1$$

$$\nu_i = \phi_1 \nu_{i-1} + \phi_2 \nu_{i-2} + \theta_1 \text{err}_{i-1} + \theta_2 \text{err}_{i-2}, \text{ for } i = 3, \ldots, T$$

$$\text{err}_i \sim \text{Normal}(0, \sigma), \text{ for } i = 1, \ldots, T$$

Here ν represents the true values of remainders (which are latent), while err represents the difference between observed values and true values, and err has a normal distribution.

Prior choice explanation: (1) σ is positive-valued, which can be modeled by the exponential distribution. The choice of $\lambda = 0.1$ is due to the same reasoning above. (2) ϕ, θ and err are real numbers that are potentially unbounded, so a normal distribution should be used. SD of ϕ and θ aims to be large due to the same reasoning before. SD of err is σ, which is already being modeled.

3.3 Posterior evaluation

As a posterior predictive check, calibration should first be examined. Specifically, a 99% credible interval is calibrated if it captures the true data 99% of the time. If it is already calibrated, then further posterior checks are not necessary. If it is not calibrated, then we need to go through the Bayesian workflow to identify the problem.

Note that time series has inherent chronological order. Therefore, it only makes sense to use past data to predict future values, but not the other way around. This means that the leave-one-out technique cannot be used to check calibration based on the training set, since, for example, the last $n-1$ data points cannot be used to predict the first data point, otherwise we are predicting the past using the future. Due to this characteristic of time series, I used the model built by the training set to predict the CPI index in the test set and compute a 99% credible interval for each data point in the test set. Then I look at the proportion of test data that is captured by its corresponding credible interval and compare this proportion with 99% to check calibration.

The following is how I got the 99% credible intervals for each data point in the test set: This is a multi-step process to obtain point forecasts, by decomposing the original time series into 3 components (trend, seasonal and remainder). As mentioned above, the seasonal component does not involve any randomness, so we can treat it as a constant yet periodic series. Two Bayesian models are built separately for the trend and remainder component, giving two sets of posterior samples. Since the point forecasts are calculated by summing these 3 components, the aggregate posterior sample for each point forecast is also the sum of the corresponding seasonal value and the two posterior samples (for trend and remainder). This is because the posterior distribution of a sum of components can be formed by summing the draws from the posterior distributions of each component. After the aggregate posterior sample is available, the 99% credible interval is obtained by extracting the 0.5% and 99.5% quantile of this aggregate posterior sample.

After computing point forecasts and their corresponding 99% credible intervals, the visualization and result summary is as follows:

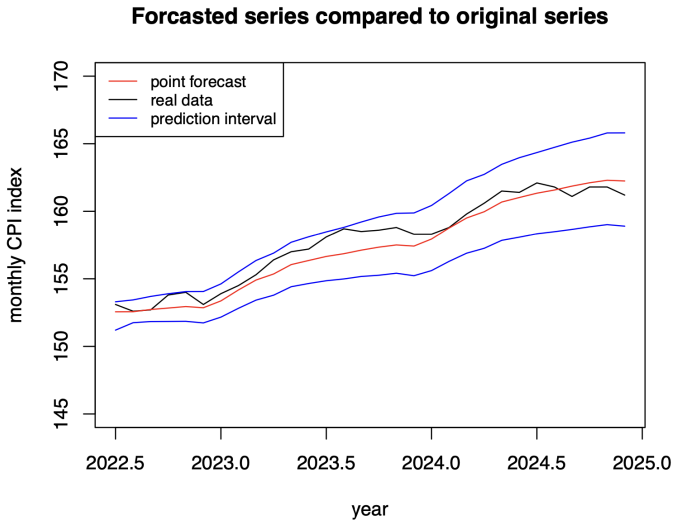


Figure 2: Calibration check

size of test set	number of data points captured by credible interval	proportion being captured
30	30	100%

As we can see, the proportion of test data captured by 99% credible interval is 100%. Since 100% > 99% and they are pretty close, we can conclude that these credible intervals are calibrated. So the posterior predictive check passes, and further examination such as diagnostics is not necessary in this case.

Apart from calibration, root mean squared error (RMSE) can also provide insights about prediction performance. In our case, RMSE turns out to be 0.84. The values of CPI index in the test set is roughly between 150 and 160, so an error of 0.84 is relatively small and tolerable. Thus our model does a decent job in prediction.

4. Discussion

The analysis has the following pros and cons:

Pros:

(1) According to Figure 2, the point forecasts follow the real data closely, exhibiting an upward trend and some cyclical variations. So the key features of the data are modeled correctly.

(2) The credible intervals capture the true data 100% of the time, showing that they are calibrated.

(3) The RMSE is 0.84, which is small compared to the scale of the data (between 150 and 160). So the prediction performance is satisfactory.

Cons:

(1) The point forecasts are smoother than real data, failing to capture the subtle fluctuations of the real data very accurately.

(2) Leave-one-out posterior predictive check is not applicable here. Instead our calibration check relies purely on the test set. The size of the test set is 30, which is not large enough. However, it cannot be much larger than 30 due to the characteristic of this CPI time series. Specifically, the pattern of the data changed in 2020 because the CPI index has increased more rapidly since 2020 than before. So we need enough data points after 2020 in the training set so as to yield a model with satisfactory predictive performance. If, however, there is no enough data after 2020 in the training set, then the model will only learn the pattern of the data before 2020. As a result, the point forecasts will be much lower than the real data in the test set. That is, our model does poorly in prediction. Due to limited data after 2020 (we have only 48 data points in total), the number of data points allocated to the test set cannot be too large. So 30 is already close to the largest number we can allocate to the test set, after taking prediction performance into account.

5. Bibliography

1. Statistics Canada. (2019, May 28). Consumer price index portal. Www.statcan.gc.ca. https://www.statcan.gc.ca/en/subjects-start/prices_and_price_indexes/consumer_price_indexes

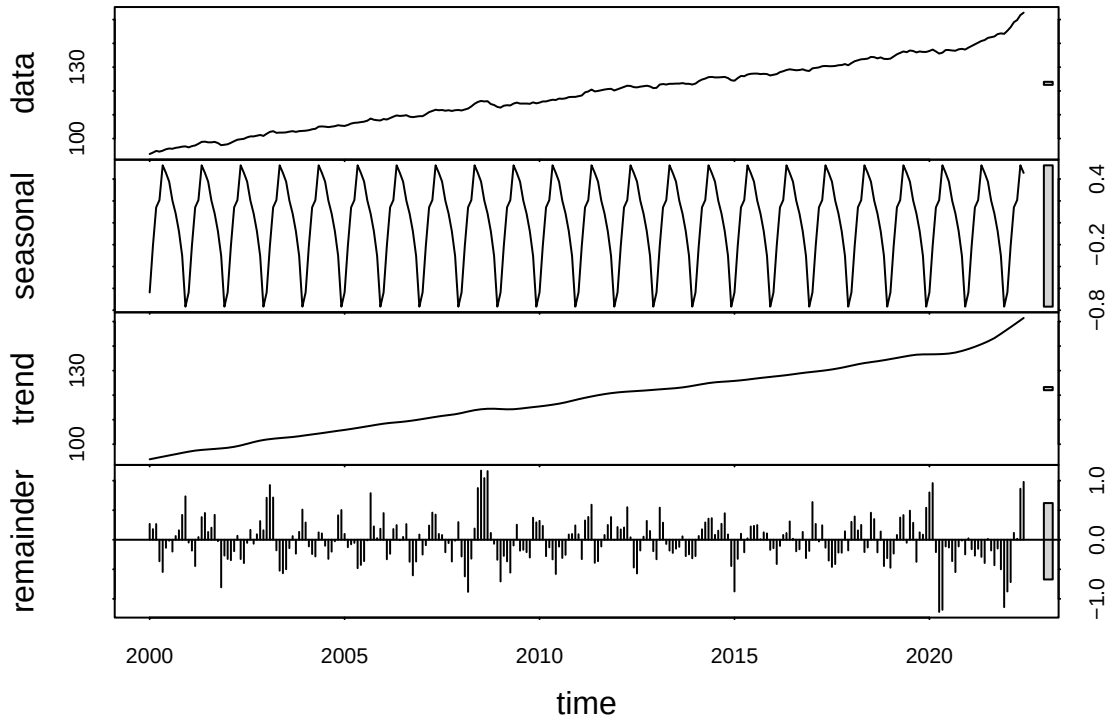
2. Cleveland, R. B., Cleveland, W. S., & Terpenning, I. (1990). STL: A seasonal-trend decomposition procedure based on loess. Journal of Official Statistics, 6(1), 3. Retrieved from https://www.proquest.com/scholarly-journals/stl-seasonal-trend-decomposition-procedure-based/docview/1266805989/se-2

3. Time-Series Models. (2025). Stan Docs. https://mc-stan.org/docs/stan-users-guide/time-series.html

6. Appendix

```r
library(tidyverse)
library(forecast)
set.seed(1)
cpi <- read.csv("../data/consumer_price_index.csv")[910:1209,]
year <- rep(2000:2024, each = 12)
month <- rep(1:12, times = 25)
cpi <- data.frame(year, month, cpi)
cpi$cpi <- as.numeric(cpi$cpi)
cpi.ts <- ts(cpi$cpi, start = c(2000,1), frequency = 12)
```

```r
set.seed(1)
training <- window(cpi.ts, c(2000,1), c(2022,6), frequency = 12)
test.ts <- window(cpi.ts, c(2022,7), c(2024,12), frequency = 12)
cpi.stl <- stl(training, s.window = "periodic")
plot(cpi.stl)
```




```r
trend <- cpi.stl$time.series[,"trend"]
seasonals <- cpi.stl$time.series[,"seasonal"]
remainders <- cpi.stl$time.series[,"remainder"]

auto.arima(remainders)
```

```
## Series: remainders
## ARIMA(2,0,2) with zero mean
##
## Coefficients:
##          ar1      ar2      ma1      ma2
##       1.3263  -0.4929  -0.6481  -0.2082
## s.e.  0.0969   0.0912   0.1097   0.1150
##
## sigma^2 = 0.08896:  log likelihood = -54.86
```

```
## AIC=119.73   AICc=119.95   BIC=137.72
```

```
# the result shows the most appropriate model for remainders is: ARMA(2,2)
```

```
require(rstan)

fit = stan(
  seed = 447,
  file = "my_stan_3.stan",
  data = list(N_train=270,
              N_test=30,
              y_train=as.numeric(trend)),
  iter = 1800,
  refresh = 0
)

samples <- extract(fit)

point_forecast <- rep(NA, 30)
lower_bound <- rep(NA, 30)
upper_bound <- rep(NA, 30)
for (i in 1:30) {
  summary <- summary(fit, pars = c("predictions"))$summary
  point_forecast[i] <- summary[i,1]
  lower_bound[i] <- summary[i,4]
  upper_bound[i] <- summary[i,8]
}
df_trend <- data.frame(1:30, point_forecast, lower_bound, upper_bound)

trend_posterior <- samples$predictions
trend_posterior <- as.data.frame(trend_posterior)
```

```
# The following is the stan code in the file "my_stan_3.stan":

data {
  int<lower=0> N_train;
  int<lower=0> N_test;
  vector[N_train] y_train;
}

parameters {
  real<lower=0> sigma1;
  real<lower=0> sigma2;
  real slope;
  real intercept;
  vector[N_train] x_train;
}

model {
  sigma1 ~ exponential(0.1);
  sigma2 ~ exponential(0.1);
  slope ~ normal(0,100);
  intercept ~ normal(0,100);

  for (i in 2:N_train) {
```

```
    x_train[i] ~ normal(intercept + slope * x_train[i-1], sigma1);
  }

  for (j in 1:N_train) {
    y_train[j] ~ normal(x_train[j], sigma2);
  }
}

generated quantities {
  vector[N_test] predictions;

  predictions[1] = normal_rng(intercept + slope * x_train[N_train], sigma1);

  for (k in 2:N_test) {
    predictions[k] = normal_rng(intercept + slope * predictions[k-1], sigma1);
  }
}
```

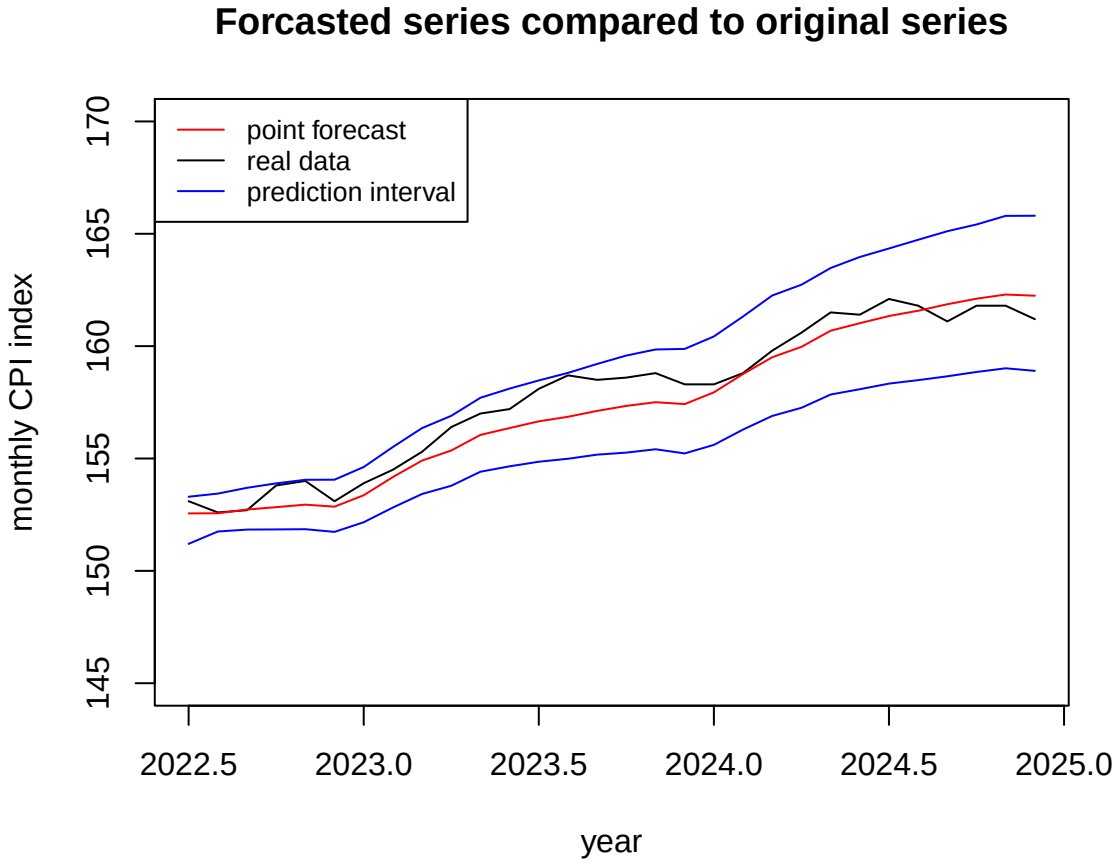
```
require(rstan)

fit = stan(
  seed = 447,
  file = "my_stan_4.stan",
  data = list(T=270,
              N_test=30,
              y=as.numeric(remainders)),
  iter = 1800,
  refresh = 0
)

samples <- extract(fit)

point_forecast <- rep(NA, 30)
lower_bound <- rep(NA, 30)
upper_bound <- rep(NA, 30)
for (i in 1:30) {
  summary <- summary(fit, pars = c("predictions"))$summary
  point_forecast[i] <- summary[i,1]
  lower_bound[i] <- summary[i,4]
  upper_bound[i] <- summary[i,8]
}
df_remainders <- data.frame(1:30, point_forecast, lower_bound, upper_bound)

remainder_posterior <- samples$predictions
remainder_posterior <- as.data.frame(remainder_posterior)
```

```
# The following is the stan code in the file "my_stan_4.stan":
# Citation: This is adapted from the stan user guide:
#           https://mc-stan.org/docs/stan-users-guide/time-series.html

data {
  int<lower=2> T;           // num observations
  int<lower=0> N_test;
  vector[T] y;              // observed outputs
```

```
}

parameters {
  vector[2] phi;              // autoregression coeff
  vector[2] theta;            // moving avg coeff
  real<lower=0> sigma;        // noise scale
}

transformed parameters {
  vector[T] err;             // error for time t
  vector[T] nu;              // prediction for time t
  nu[1] = 0;                 // assume err[0] == 0
  err[1] = y[1] - nu[1];
  nu[2] = phi[1] * nu[1] + theta[1] * err[1];
  err[2] = y[2] - nu[2];

  for (t in 3:T) {
    nu[t] = phi[1] * nu[t - 1] + phi[2] * nu[t - 2] + theta[1] * err[t - 1] + theta[2] * err[t - 2];
    err[t] = y[t] - nu[t];
  }
}

model {
  phi ~ normal(0, 100);         // priors
  theta ~ normal(0, 100);
  sigma ~ exponential(0.1);
  err ~ normal(0, sigma);      // error model
}

generated quantities {
  vector[N_test] predictions;

  predictions[1] = phi[1] * nu[T] + phi[2] * nu[T-1] + theta[1] * err[T] + theta[2] * err[T-1];

  predictions[2] = phi[1] * predictions[1] + phi[2] * nu[T] + theta[2] * err[T];

  for (k in 3:N_test) {
    predictions[k] = phi[1] * predictions[k-1] + phi[2] * predictions[k-2];
  }
}
```

```
set.seed(1)
trend_estimates <- ts(df_trend$point_forecast, start = c(2022,7), frequency = 12)
seasonal_estimates <- window(seasonals, start = c(2000,7), end = c(2002,12))
seasonal_estimates <- ts(as.numeric(seasonal_estimates), start = c(2022,7), frequency = 12)
remainder_estimates <- ts(df_remainders$point_forecast, start = c(2022,7), frequency = 12)

posterior <- trend_posterior + remainder_posterior
for (i in 1:30) {
  posterior[,i] <- posterior[,i] + as.numeric(seasonal_estimates)[i]
}
quantiles <- apply(posterior[],2,quantile,probs=c(0.005,0.995))
lower.bound <- quantiles[1,]
upper.bound <- quantiles[2,]
```

```r
point.forcast.ts <- trend_estimates + seasonal_estimates + remainder_estimates
lower.bound.ts <- ts(as.vector(lower.bound), start = c(2022,7), frequency = 12)
upper.bound.ts <- ts(as.vector(upper.bound), start = c(2022,7), frequency = 12)
ts.plot(test.ts, point.forcast.ts, lower.bound.ts, upper.bound.ts,
        gpars = list(xlab = "year",
                     ylab = "monthly CPI index",
                     main = "Forcasted series compared to original series",
                     col = c("black", "red", "blue", "blue")
                     ),
        ylim = c(145, 170)
        )
legend("topleft", legend = c("point forecast", "real data", "prediction interval"),
col = c("red", "black","blue"),lty = c(1,1,1), cex=0.8)
```




```r
set.seed(1)
warning <- 0
real_data <- as.vector(test.ts)
for (i in 1:30) {
  if (real_data[i] < lower.bound[i] | real_data[i] > upper.bound[i]) {
    warning <- warning + 1
  }
}
proportion_captured <- 1-warning/30

cat("The number of test data outside its corresponding prediction interval is: ",
    warning, "\n")
```

```
## The number of test data outside its corresponding prediction interval is:  0
```

```
cat("Proportion of test data captured by corresponding prediction interval is: ",
    proportion_captured, "\n")
```

```
## Proportion of test data captured by corresponding prediction interval is:  1
```

```
rmse <- sqrt(mean((as.vector(point.forcast.ts) - real_data)^2))

cat("RMSE equals: ", rmse)
```

```
## RMSE equals:  0.8396362
```