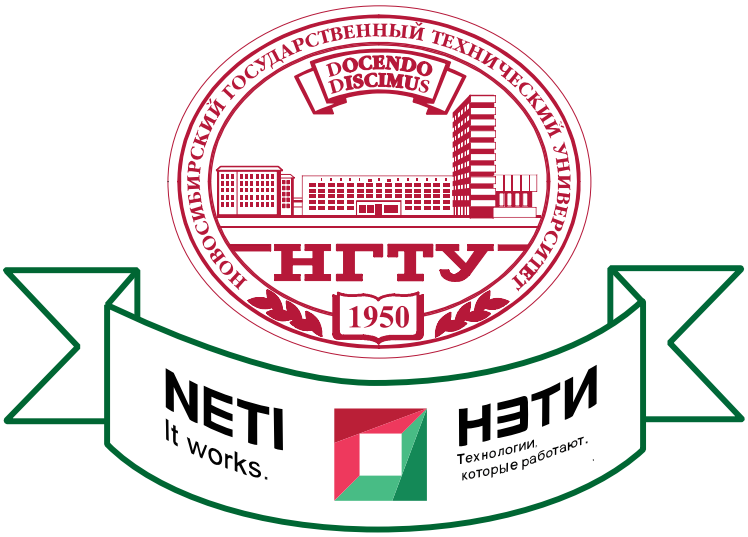
Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«Новосибирский государственный технический университет»

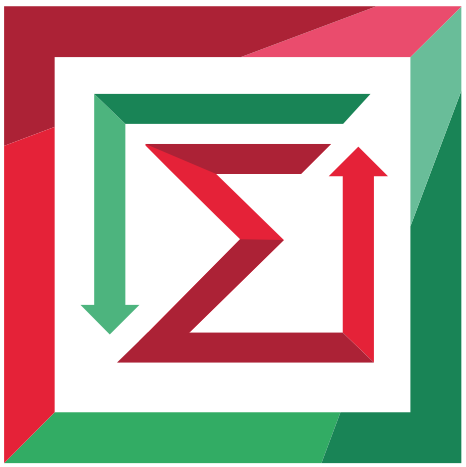


Теоретической и прикладной информатики

Лабораторная работа № 3

по дисциплине «Компьютерное моделирование»

Построение прогноза регрессионных, авторегрессионных моделей и моделей в пространстве состояний.



Факультет:	ПМИ
Группа:	ПМИ-02
Студент:	Сидоров Даниил,
	Дюков Богдан
Преподаватель:	Карманов Виталий Сергеевич

Новосибирск

2026

1. Исходные данные

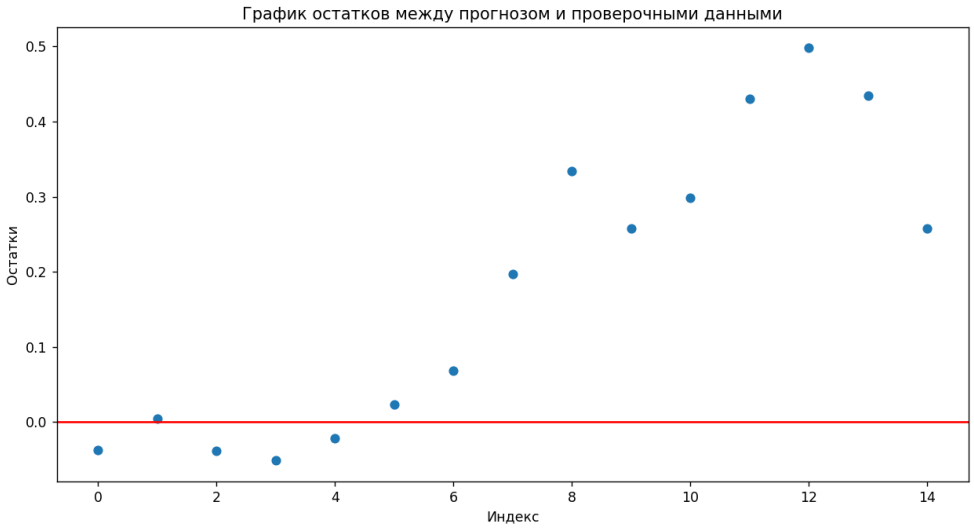
Временные ряды x(t), t=0,1,2,…,n; y(t), t=0,1,2,…,m, описывающие динамику двух различных процессов (из лабораторной работы №1).

Данные курса дирхам были поделены на две части. Большая часть, равная 95%, нужна для обучения, а остальные 5% будут использованы для проверки прогноза. Аналогично поступили с предложенными данными-1, за исключением того, что две части поделены на 90% и 10%, так как объем выборки мал.

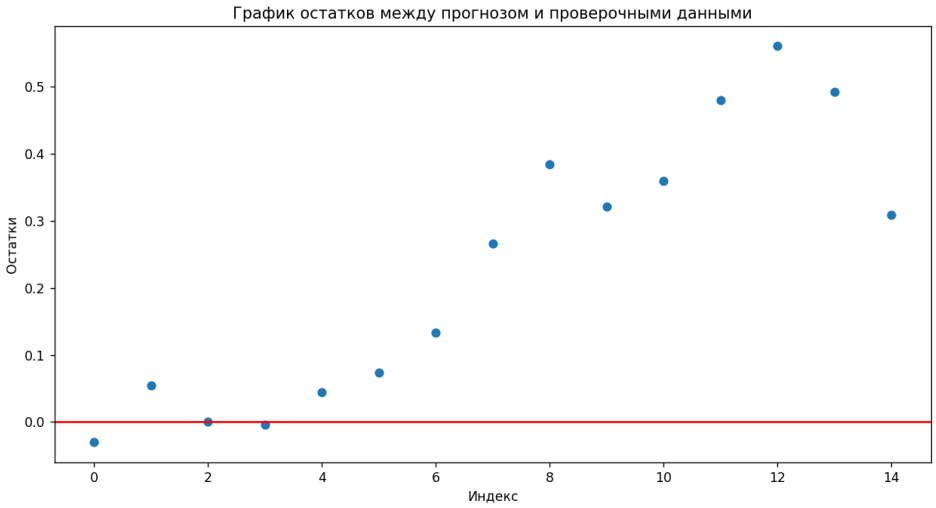
2. Описание выполненных действий

При выборе оптимальных параметров моделей, будем ориентироваться на полученные результаты во второй лабораторной работе.

Прогноз ARIMA курс дирхам

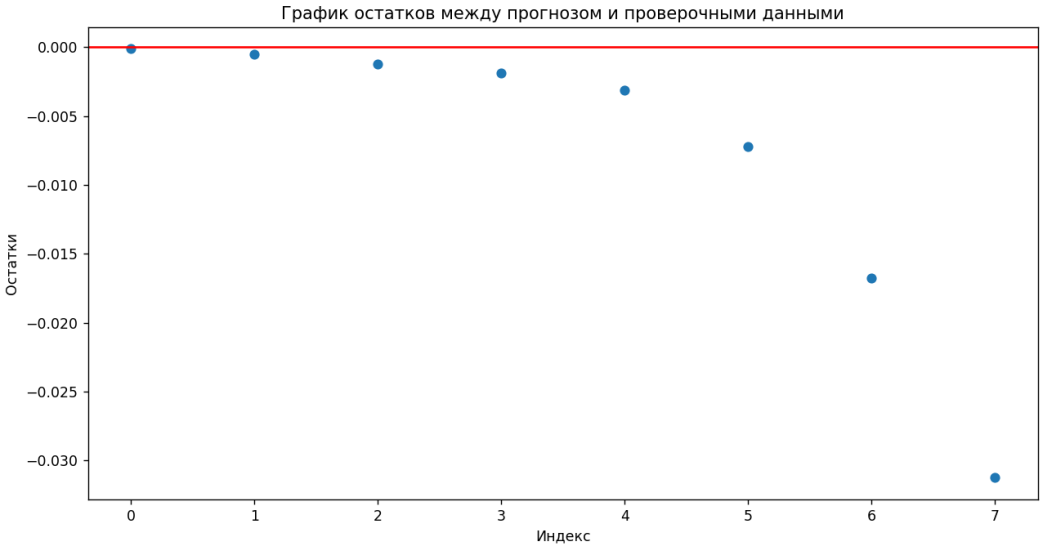


RMSE = 1.078

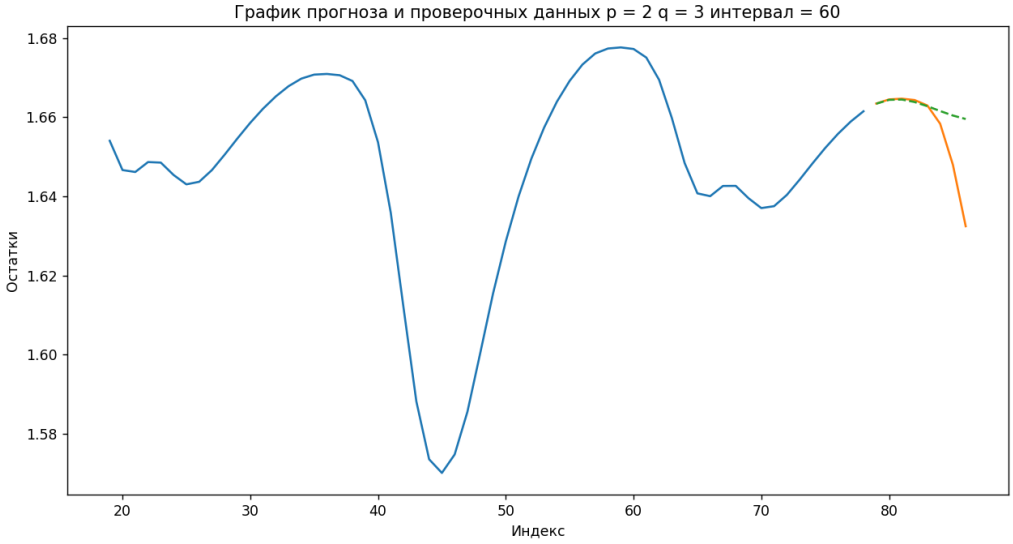


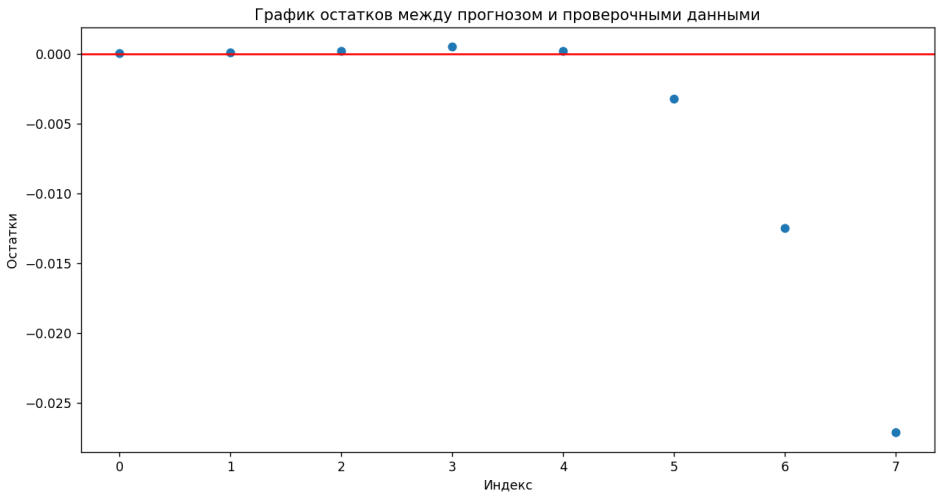
RMSE = 1.063

Прогноз ARIMA данные-1



RMSE = 0.11

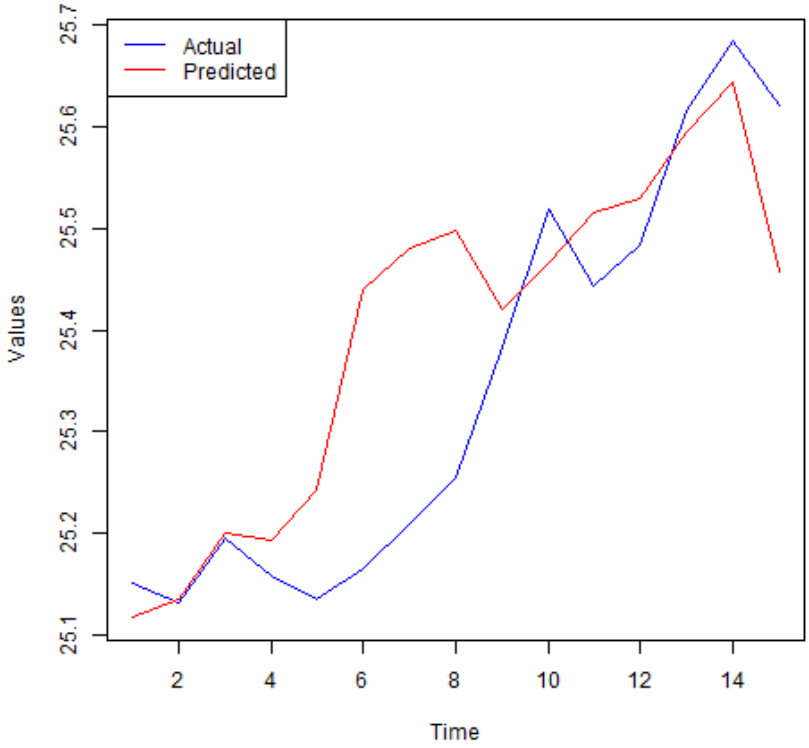


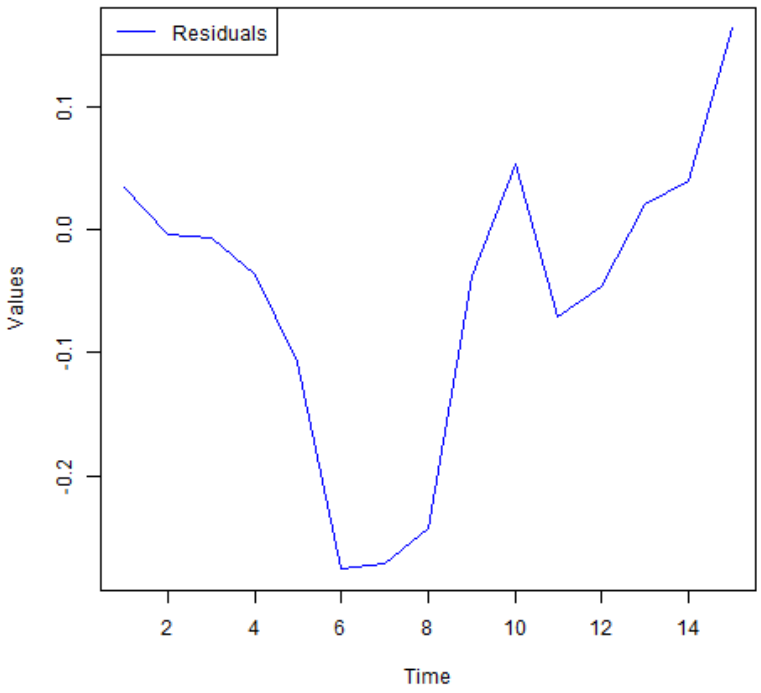


RMSE = 0.009

Прогноз BSTS курс дирхам

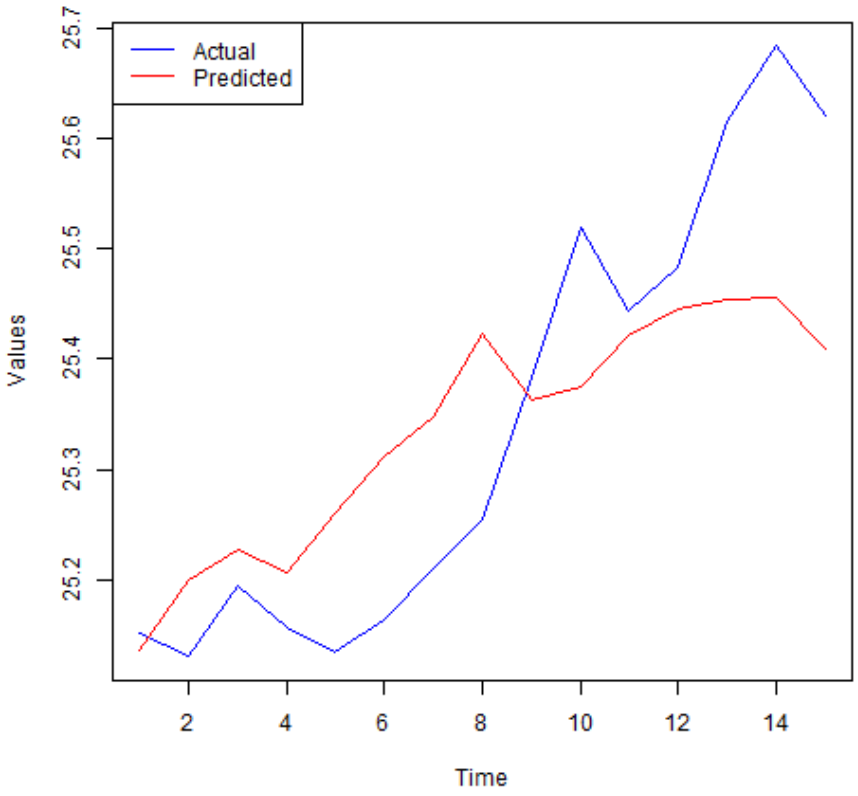
Интервал 210, число итераций 500, сезонность 24

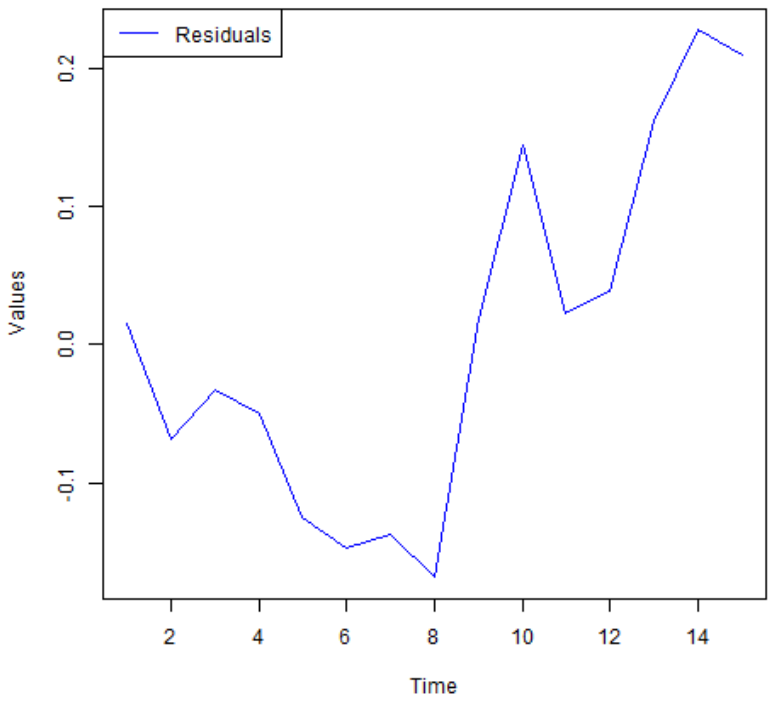




RMSE = 0.173

Интервал 285, число итераций 500, сезонность 6

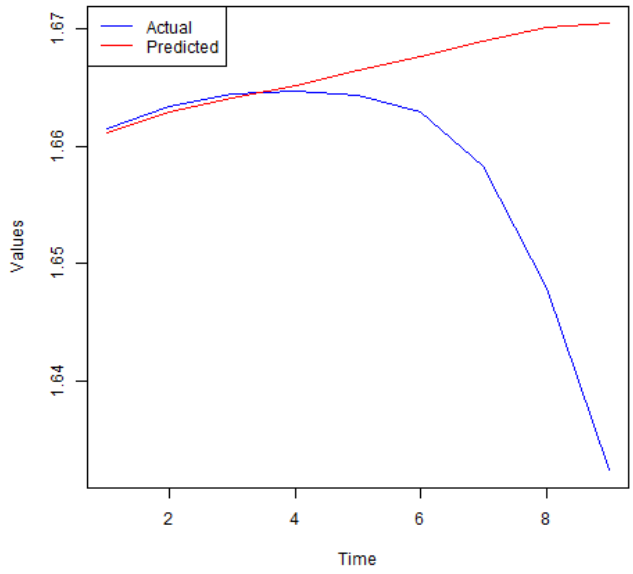


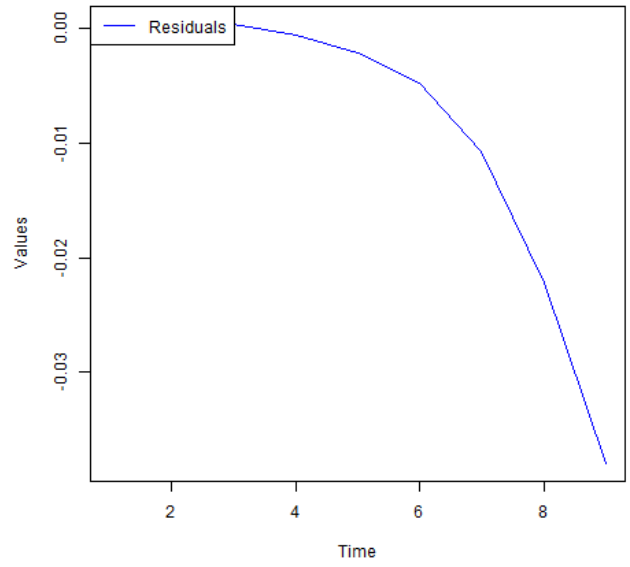


RMSE = 0.201

Прогноз BSTS данные-1

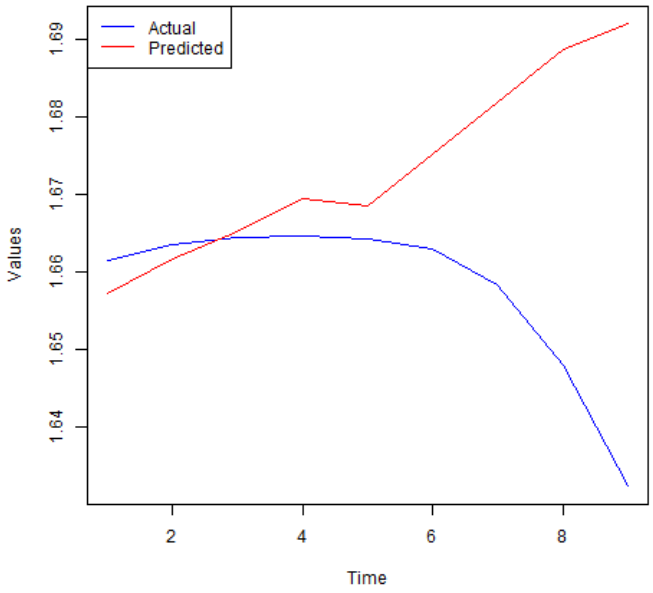
Интервал 78, число итераций 1500, сезонность 24

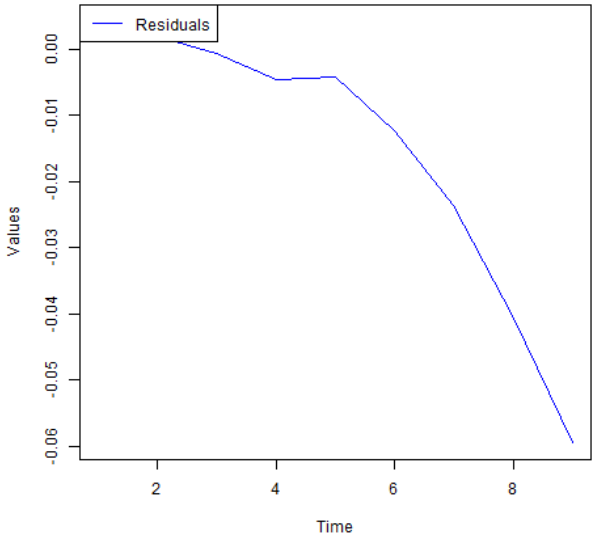




RMSE = 0.023

Интервал 20, число итераций 500, сезонность 24

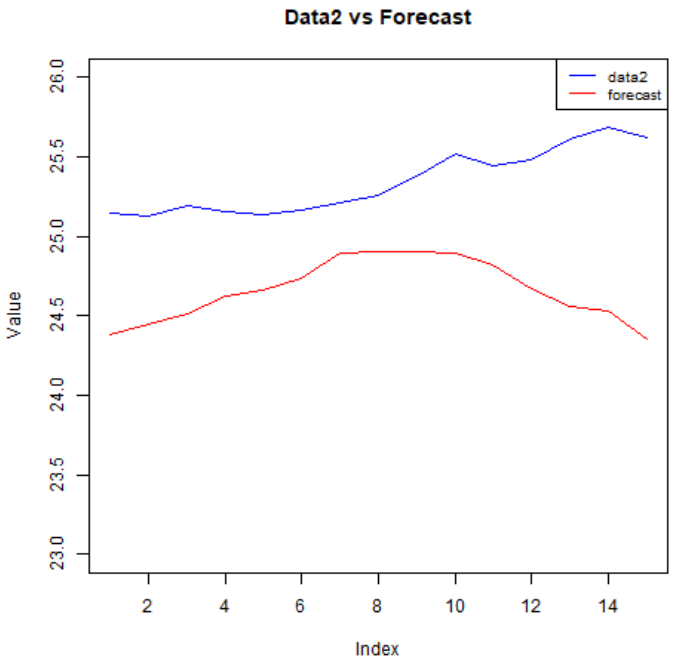


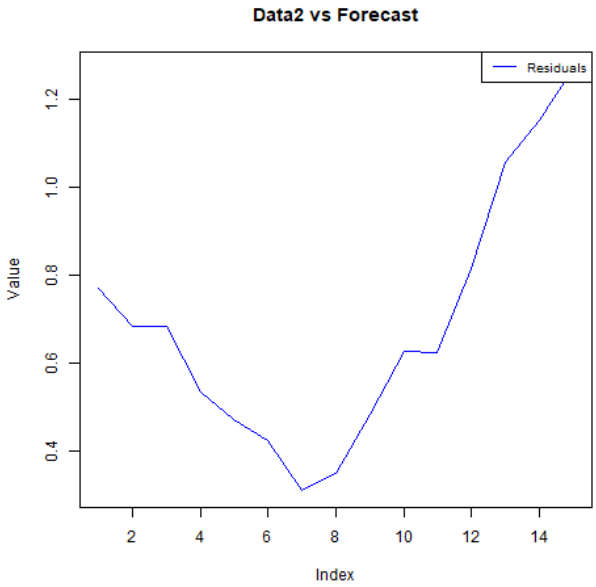


RMSE = 0.056

Прогноз аддитивной нелинейной регрессионной модели

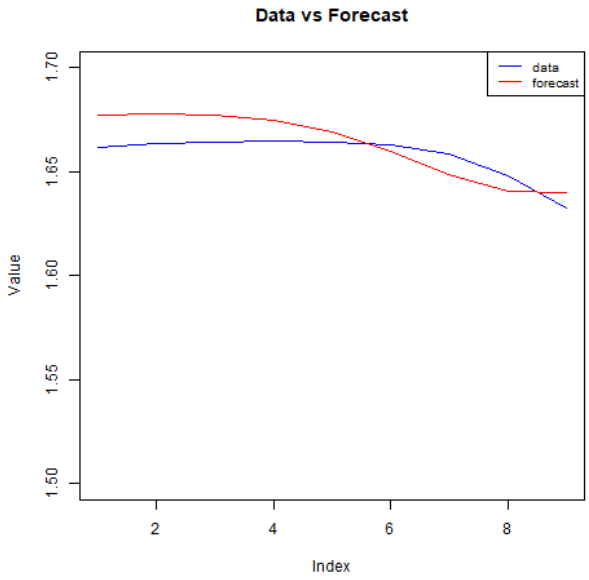
Прогноз Prophet курс дирхам

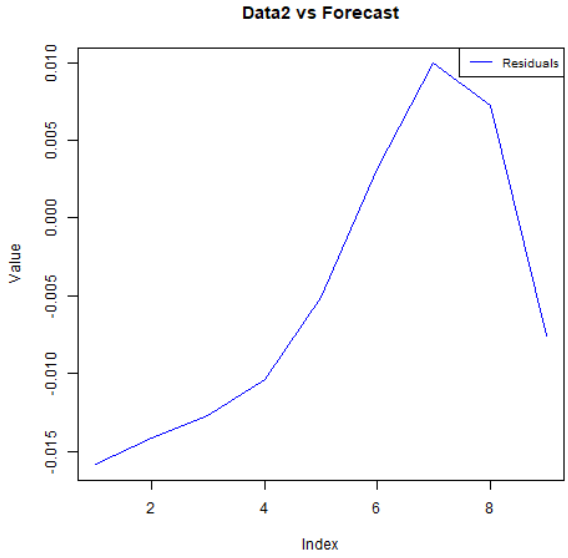




RMSE = 0.544

Прогноз Prophet данные-1





RMSE = 0.0013

Прогноз SSA

Прогноз SSA курс валют

r = 15, L = 50


SSA прогноз
Модель
Выборка
Прогноз
Истинные данные

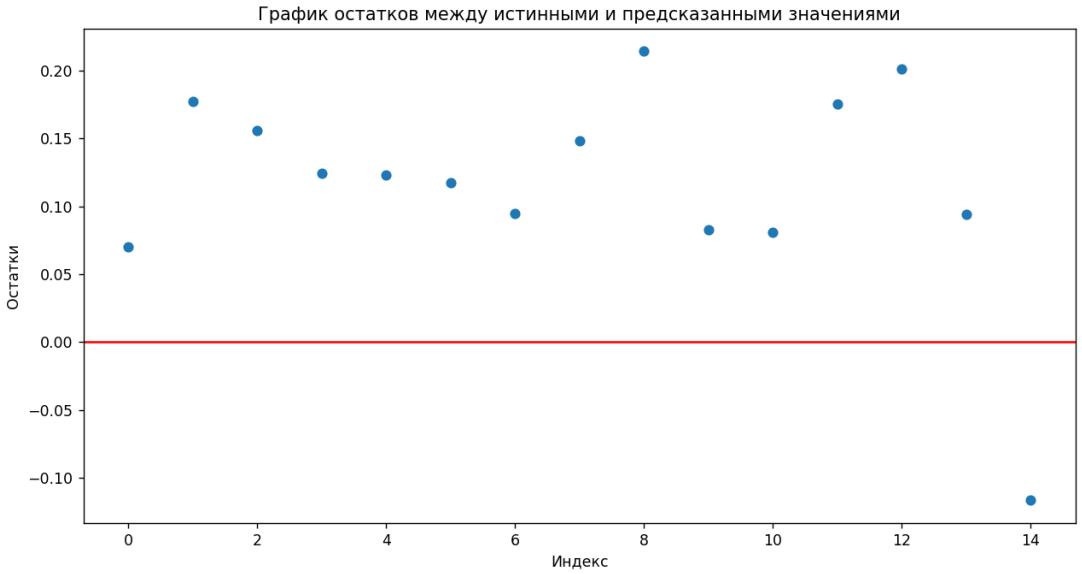


График остатков между истинными и предсказанными значениями
Остатки
Индекс

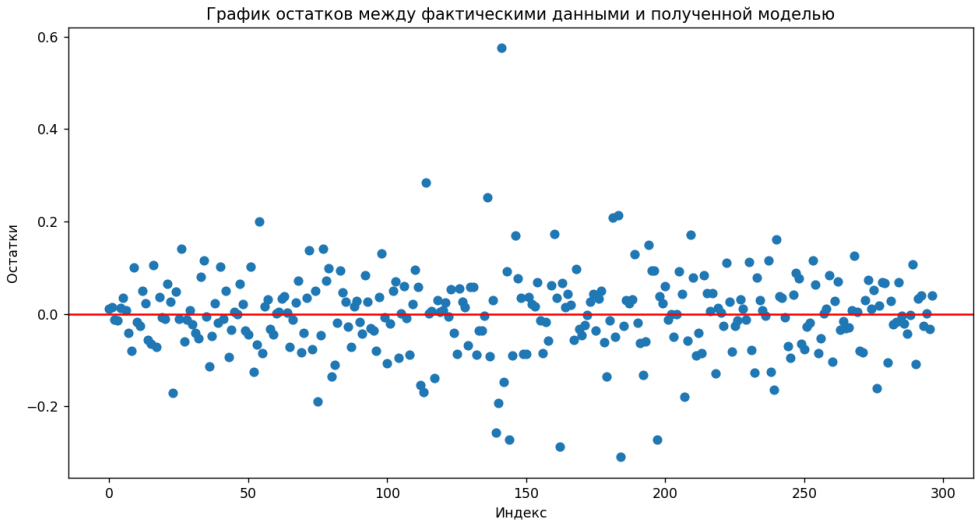
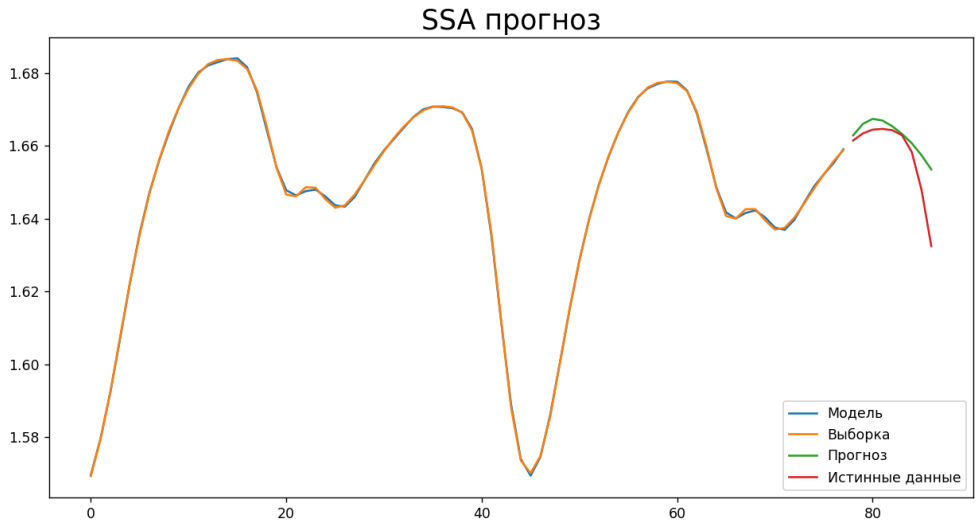


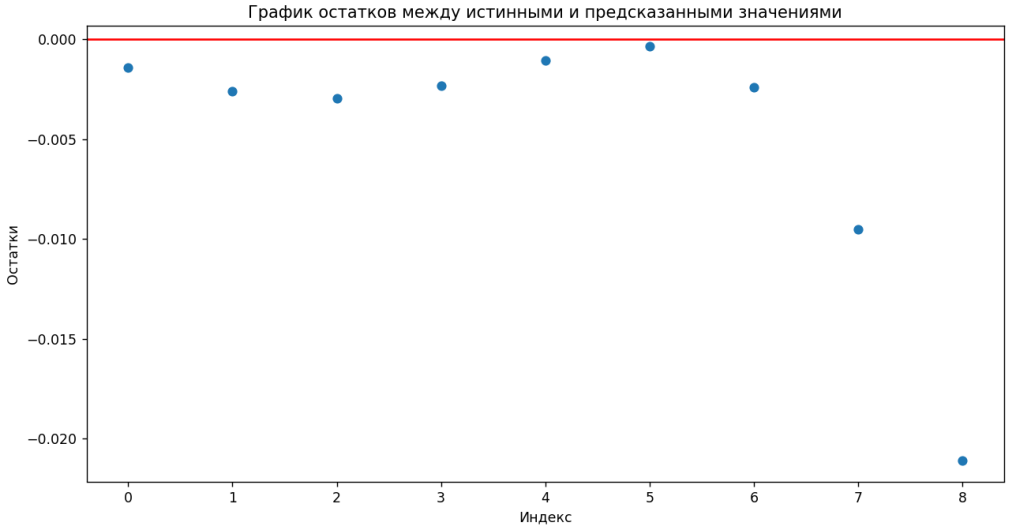
График остатков между фактическими данными и полученной моделью
Остатки
Индекс

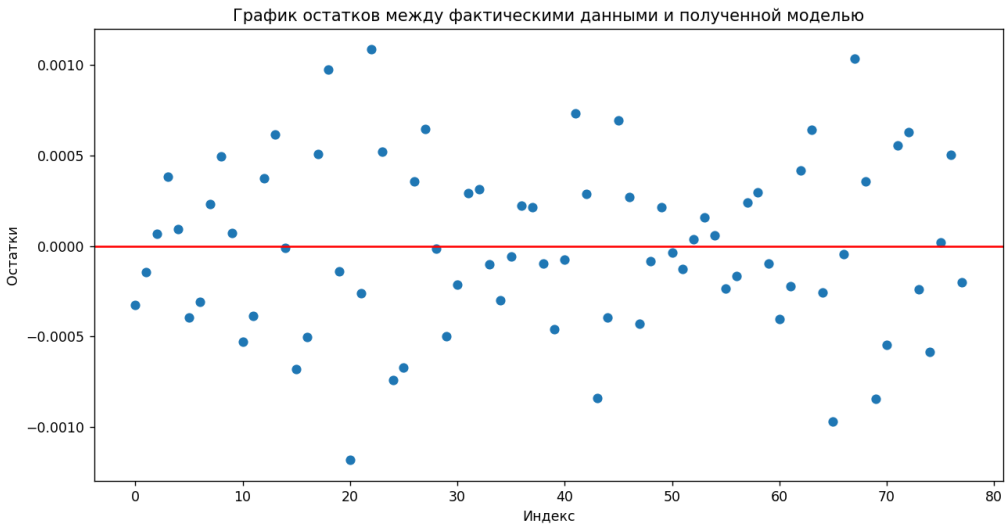
RMSE = 0.134

Прогноз SSA данные-1

r = 10, L = 55







RMSE = 0.0079

3. Вывод

Подведем итог в виде сравнения всех прогнозов моделей.

Лучше всех себя проявил метод SSA, который довольно точно совершал прогноз для обеих выборок с минимальным RMSE.

Модель ARIMA неплохо показала себя для данных-1, но для курса валют оказалась неэффективной.

Метод BSTS хорошо предсказал курс валют, но не совсем успешно справился с данными-1.

Prophet, как и SSA, продемонстрировала хороший прогноз, но с большим RMSE в сравнении с тем же SSA.

4. Листинг программы

ARIMA:

```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from itertools import product
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import warnings

# Подавить предупреждения
warnings.filterwarnings("ignore")

# Загрузка данных из файла Excel
data = pd.read_excel("data-1.xlsx")
time_series = data["smoothed_data"]
len_data = len(time_series)
#interval_lengths = [60, 120, 180, 228]  # Пример значений длин интервалов
interval_lengths = [20, 40, 60, 67]  # Пример значений длин интервалов


def plot_comparison(data, forecast, order, interval):
    plt.figure(figsize=(10, 6))
    plt.plot(data, label='Actual data')
    plt.plot(forecast, label='Forecast', linestyle='--')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.title(f'Comparison: Order={order}, Interval={interval}')
    plt.legend()

plt.savefig(f'graph/Order_{order[0]}_{order[2]}_Interval_{interval}_len_{len_dat
a}.png')
    plt.close()



# Функция для определения оптимальных параметров модели ARIMA с использованием
критерия Акаике
```

```
def find_best_arima_params(series, p_values, d_values, q_values):
    best_aic = float("inf")
    best_params = None

    for p, d, q in product(p_values, d_values, q_values):
        try:
            model = ARIMA(series, order=(p, d, q))
            results = model.fit()

            # Прогноз на следующие 20 значений
            forecast = results.forecast(steps=20)

            aic = results.aic
            #residuals = results.resid

            #mse = np.std(residuals)
            #print(residuals)

            if aic < best_aic:
                best_aic = aic
                best_params = (p, d, q, best_aic)
                best_forecast = forecast
        except:
            continue

    return best_params, best_forecast


# Определение оптимальных параметров для различных длин мерных интервалов
results = []

for length in interval_lengths:
    #print(len_data, length)
    interval_series = time_series[-(length + 20):(len_data - 20)]  # Выбираем
первые length значений
    p_values = range(1, 5)  # Пример значений для p
    d_values = [1]  # Пример значений для d
    q_values = range(1, 5)  # Пример значений для q

    best_params, forecast = find_best_arima_params(interval_series, p_values,
d_values, q_values)

    mse = mean_squared_error(time_series[-20:], forecast)

    plot_comparison(data[-20:], forecast, best_params, length)

    results.append((length, best_params, mse))

# Вывод результатов
print("Длина интервала | Оптимальные параметры (p, d, q, aic, mse)")
for length, params, mse in results:
    print(f"{length:<15} | {params} | {mse}")
```

```
# Функция для настройки и оценки модели ARIMA
def evaluate_arima_model(order, data):
    p, d, q = order
    try:
        # Построение модели ARIMA
        model = ARIMA(data, order=order)
        model_fit = model.fit()

        # Прогноз на следующие 20 значений
        forecast = model_fit.forecast(steps=20)

        # Получение прогнозов
        #residuals = model_fit.resid
        #mse = np.std(residuals)
        return forecast
    except:
        return None

# Функция для поиска оптимальных параметров модели ARIMA
def find_best_arima_parameters(data, intervals):
    best_params = []
    params = []
    for interval in intervals:
        # Создаем список комбинаций параметров
        p_values = range(1, 5)
        d_values = [1]  # Потому что d=1 для нестационарных данных
        q_values = range(1, 5)
        orders = product(p_values, d_values, q_values)

        # Находим оптимальные параметры на основе критерия Акаике
        best_mse = float('inf')
        best_order = None
        best_forecast = None
        for order in orders:
            forecast = evaluate_arima_model(order, data[-(interval +
20):len_data - 20])
            mse = mean_squared_error(data[-20:], forecast)
            params.append({'Interval Length': interval,
                           'p': order[0],
                           'q': order[2],
                           'MSE': mse})
            if mse is not None and mse < best_mse:
                best_mse = mse
                best_order = order
                best_forecast = forecast


        best_params.append({'Interval Length': interval,
                            'p': best_order[0],
                            'q': best_order[2],
                            'MSE': best_mse})
        plot_comparison(data[-20:], best_forecast, best_order, interval)


    return params, best_params


# Находим оптимальные параметры для каждой длины мерного интервала
params, best_params = find_best_arima_parameters(time_series, interval_lengths)
```

```
# Создаем DataFrame и выводим результаты
params = pd.DataFrame(params)
# Записываем результаты в файл Excel
params.to_excel('result.xlsx', index=False)
best_params = pd.DataFrame(best_params)
print(params)
print(best_params)
```

Тренды и сезоны:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy.polynomial import chebyshev
from scipy.optimize import curve_fit
from scipy.fft import fft
# Проверка нормальности остатков
from scipy.stats import shapiro
import statsmodels.api as sm
from statsmodels.tsa.ar_model import AutoReg
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import acf
import warnings

# Подавить предупреждения
warnings.filterwarnings("ignore")

# Загрузка данных из файла Excel
data = pd.read_excel("data.xlsx")
y1 = data["smoothed_curs"]#.tail(60)
len_data = len(y1)
y = y1[:(len_data-20)]
true_y = y1[-20:]
x = np.arange(len(y))
x1 = np.arange(len(y1))


# Степени полиномов
degrees = [1, 2, 3, 4]
cheb_degrees = [3]  # Степени многочленов Чебышева
poly_trend = 0
cheby_trend = 0
# Визуализация результатов
plt.figure(figsize=(10, 6))
plt.plot(x, y, label='Original data')

# Полиномиальные тренды
for degree in degrees:
    # Подготовка данных для полиномиальной регрессии
    X = np.vander(x, degree + 1)  # Матрица Вандермонда
    coefficients = np.linalg.lstsq(X, y, rcond=None)[0]

    # Генерация значений для тренда
    trend = np.dot(X, coefficients)
    poly_trend = trend
    # Построение тренда
    plt.plot(x, trend, label=f'Polynomial Trend (degree={degree})')
```

```
# Многочлены Чебышева
for cheb_degree in cheb_degrees:
    cheb_coeffs = chebyshev.chebfit(x, y, cheb_degree)
    cheb_trend = chebyshev.chebval(x, cheb_coeffs)
    cheby_trend = cheb_trend
    # Построение тренда многочленов Чебышева
    plt.plot(x, cheb_trend, label=f'Chebyshev Polynomial Trend
(degree={cheb_degree})')

plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Trend Extraction with Polynomial Regression and Chebyshev
Polynomials')
plt.legend()
plt.grid(True)
plt.show()



# Определение переменных
y = y.to_numpy()  # Преобразуем столбец данных в массив numpy
n = len(y)  # Длина временного ряда
dt = 1  # Шаг времени (предполагаем, что данные измеряются с постоянным
интервалом)

# Выполнение преобразования Фурье
frequencies = np.fft.fftfreq(n, dt)
fft_values = fft(y)

# Находим амплитуды и фазы
amplitudes = np.abs(fft_values) / n
phases = np.angle(fft_values)

# Определение частоты сезонной компоненты (гармонической составляющей)
seasonal_frequency_index = np.argmax(amplitudes[1:]) + 1
seasonal_frequency = frequencies[seasonal_frequency_index]
seasonal_period = 1 / seasonal_frequency

# Визуализация результатов
plt.figure(figsize=(10, 6))
plt.stem(frequencies[:n//2], amplitudes[:n//2])
plt.ylim(bottom=0, top=0.5)  # Устанавливаем нижний и верхний пределы для оси y
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('Fourier Transform')
plt.grid(True)
plt.show()

print("Seasonal component frequency:", seasonal_frequency)
print("Seasonal component period:", seasonal_period)



# Выделение сезонной компоненты
seasonal_component = amplitudes[seasonal_frequency_index] * np.sin(2 * np.pi *
seasonal_frequency * np.arange(n) + phases[seasonal_frequency_index])

# Визуализация сезонной компоненты
plt.figure(figsize=(10, 6))
```

```
plt.plot(np.arange(n), seasonal_component, label='Seasonal Component',
color='red')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Seasonal Component Visualization')
plt.legend()
plt.grid(True)
plt.show()



trend_plus_seasonal = poly_trend + seasonal_component
#trend_plus_seasonal = cheby_trend + seasonal_component
residuals = y - trend_plus_seasonal


#plot_acf(residuals, lags=len(y)-1)
#plt.title('Autocorrelation Function (ACF) of Residuals')
#plt.show()

#acf_values = acf(residuals, nlags=len(y)-1, fft=False)

#print(acf_values)

#residuals = residuals + acf_values
#trend_plus_seasonal = trend_plus_seasonal + acf_values



mse = np.std(residuals)
stat, p = shapiro(residuals)
alpha = 0.05
if p > alpha:
    print('Остатки имеют нормальное распределение, mse:', mse)
else:
    print('Остатки не имеют нормальное распределение, mse:', mse)

# Визуализация остатков
plt.figure(figsize=(10, 6))
plt.plot(x, residuals, label='Residuals', color='green')
plt.axhline(0, color='red', linestyle='--')  # Горизонтальная линия на уровне
нуля
plt.xlabel('Time')
plt.ylabel('Residuals')
plt.title('Residuals Visualization')
plt.legend()
plt.grid(True)
plt.show()



# Визуализация исходных данных и trend_plus_seasonal
plt.figure(figsize=(10, 6))
plt.plot(x, y, label='Original data')
plt.plot(x, trend_plus_seasonal, label='Trend + Seasonal', color='orange')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Original Data vs Trend + Seasonal')
```

```
plt.legend()
plt.grid(True)
plt.show()


# Генерация временных шагов для предсказания следующих 20 значений
future_time_steps = np.arange(len_data - 20, len_data)

# Предсказание следующих 20 значений с использованием тренда плюс сезонной
компоненты
future_predictions = trend_plus_seasonal[-1] * np.ones(20)  # Первое
предсказанное значение равно последнему известному

# Визуализация предсказанных значений
plt.figure(figsize=(10, 6))
plt.plot(x1, y1, label='Original data')
plt.plot(x, trend_plus_seasonal, label='Trend + Seasonal', color='orange')
plt.plot(future_time_steps, future_predictions, label='Future Predictions',
color='green', linestyle='--')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Original Data vs Trend + Seasonal with Future Predictions')
plt.legend()
plt.grid(True)
plt.show()

import numpy as np
import pandas as pd
from scipy import fftpack
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from scipy.stats import normaltest
from sklearn.linear_model import Ridge
from scipy.fft import fft
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from scipy.stats import shapiro
from numpy.polynomial import Chebyshev

# Загрузка данных
data = pd.read_excel("data-1.xlsx")
y = data["smoothed_data"].values
interval = 60
test = y[len(y)-20:]
y = y[-(interval+20):]
train = y[:len(y)-20]
# Используем только первые N-20 значений для обучения модели
N = len(y)


# Выделение трендовой составляющей
X_train = np.arange(len(train)).reshape(-1, 1)
model = make_pipeline(PolynomialFeatures(1), LinearRegression())  #
Инициализация model
model.fit(X_train, train)
trend_train = model.predict(X_train)
```

```
# График трендовой составляющей
plt.figure(figsize=(12, 8))
plt.plot(trend_train, label='Трендовая составляющая')
plt.legend(loc='upper left')
plt.show()

# Выделение сезонной составляющей
#y_detrend_train = train - trend_train
#frequencies = fftpack.fftfreq(len(train))
#positive_freqs = frequencies[frequencies > 0]
#powers = np.abs(fftpack.fft(train))[frequencies > 0]
#peak_freq = positive_freqs[powers.argmax()]
#seasonal_train = np.sin(2 * np.pi * peak_freq * X_train.squeeze())* peak_freq



# Определение переменных
n = len(train)  # Длина временного ряда
print(N)
dt = 1  # Шаг времени (предполагаем, что данные измеряются с постоянным
интервалом)

# Выполнение преобразования Фурье
frequencies = np.fft.fftfreq(n, dt)
fft_values = fft(train)

# Находим амплитуды и фазы
amplitudes = np.abs(fft_values) / n
phases = np.angle(fft_values)

# Определение частоты сезонной компоненты (гармонической составляющей)
seasonal_frequency_index = np.argmax(amplitudes[1:]) + 1
seasonal_frequency = frequencies[seasonal_frequency_index]
seasonal_period = 1 / seasonal_frequency

# Визуализация результатов
plt.figure(figsize=(10, 6))
plt.stem(frequencies[:n//2], amplitudes[:n//2])
plt.ylim(bottom=0, top=0.5)  # Устанавливаем нижний и верхний пределы для оси y
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.title('Fourier Transform')
plt.grid(True)
plt.show()

print("Seasonal component frequency:", seasonal_frequency)
print("Seasonal component period:", seasonal_period)


# Выделение сезонной компоненты
seasonal_train = amplitudes[seasonal_frequency_index] * np.sin(2 * np.pi *
seasonal_frequency * np.arange(n) + phases[seasonal_frequency_index])



# График сезонной составляющей
plt.figure(figsize=(12, 8))
plt.plot(seasonal_train, label='Сезонная составляющая')
```

```
plt.legend(loc='upper left')
plt.show()

# Выделение остаточной составляющей
residual_train = train - trend_train - seasonal_train

mse = np.std(residual_train)
stat, p = shapiro(residual_train)
alpha = 0.1
if p > alpha:
    print('Остатки имеют нормальное распределение, mse:', mse)
else:
    print('Остатки не имеют нормальное распределение, mse:', mse)

# График остаточной составляющей
plt.figure(figsize=(12, 8))
plt.plot(residual_train, label='Остаточная составляющая')
plt.legend(loc='upper left')
plt.show()

# График сравнения истинных данных с суммой составляющих
plt.figure(figsize=(12, 8))
plt.plot(train, label='Истинные данные')
plt.plot(trend_train + seasonal_train, label='Сумма составляющих')
#plt.plot(trend_train, label='Сумма составляющих')
plt.legend(loc='upper left')
plt.show()

# Предсказание следующих 20 значений
X_test = np.arange(len(train), N).reshape(-1, 1)


# Продолжение сезонной составляющей в будущее
future_time_steps = 20  # Количество временных шагов для предсказания
future_seasonal_train = (amplitudes[seasonal_frequency_index] *
                         np.sin(2 * np.pi * seasonal_frequency * (N +
np.arange(future_time_steps))
                                                                  +
phases[seasonal_frequency_index]))

y_pred = model.predict(X_test) + future_seasonal_train

# Сравнение истинных и предсказанных значений
mse = mean_squared_error(test, y_pred)
print(f"MSE: {mse}")


# График истинных и предсказанных значений
plt.figure(figsize=(12, 8))
plt.plot(np.arange(N-20, N), test, label='Истинные значения')
plt.plot(np.arange(N-20, N), y_pred, label='Предсказанные значения')
plt.legend()
plt.show()
```

BSTS:

```
# Загрузка необходимых библиотек
library(bsts)
library(readxl)
```

```
library(dplyr)
library(grDevices)

# Загрузим данные
#data <- read_excel("data.xlsx", col_names = FALSE, range = "A2:A88")$...1
data <- read_excel("data.xlsx", col_names = FALSE, range = "B2:B249")$...1

evaluate_bsts_model <- function(data, niter, nseasons, horizon, data2) {
  # Построение модели BSTS
  ss <- AddLocalLinearTrend(list(), data)
  ss <- AddSeasonal(ss, data, nseasons = nseasons)
  bsts.model <- bsts(data, state.specification = ss, niter = niter)



  # Предсказание
  pred <- predict.bsts(bsts.model, horizon = horizon, level = 0.95)
  # Создание имени файла
  filename <- paste0("plot_", length(data), "_", niter_values[i], "_",
nseasons_values[j], ".png")

  # Сохранение графика
  png(filename)

  # Построение графика
  plot(data2, type = "l", col = "blue", ylim = c(min(data2, pred$mean),
max(data2, pred$mean)), ylab = "Values", xlab = "Time")
  lines(pred$mean, col = "red")
  legend("topleft", legend = c("Actual", "Predicted"), col = c("blue", "red"),
lty = 1)



  # Clear the Plot Window
  dev.off()

  # Вычисление MSE
  mse <- mean((data2 - pred$mean)^2)

  return(mse)
}

# Различные значения параметров
horizon <- 20
niter_values <- c(500, 1000, 1500)  # Различные значения niter
nseasons_values <- c(6, 12, 24)      # Различные значения nseasons
#interval_values <- c(20, 40, 60, 87 - horizon)  # Различные длины мерных
интервалов
interval_values <- c(60, 120, 180, 248 - horizon)  # Различные длины мерных
интервалов
data_length = length(data)

# Создание матрицы для сохранения результатов
results <- array(NA, dim = c(length(niter_values), length(nseasons_values),
length(interval_values)),
                 dimnames = list(niter_values, nseasons_values,
interval_values))

# Оценка точности для различных значений параметров
```

```
for (i in seq_along(niter_values)) {
  for (j in seq_along(nseasons_values)) {
    for (k in seq_along(interval_values)) {

      # Вычисление начального индекса для первого аргумента функции
evaluate_bsts_model
      start_index <- data_length - interval_values[k] - horizon + 1
      # Вычисление конечного индекса для первого аргумента функции
evaluate_bsts_model
      end_index <- data_length - horizon

      # Вычисление начального индекса для последнего аргумента функции
evaluate_bsts_model
      start_index_last_arg <- data_length - horizon + 1
      # Вычисление конечного индекса для последнего аргумента функции
evaluate_bsts_model
      end_index_last_arg <- data_length


      mse <- evaluate_bsts_model(data[start_index:end_index], niter_values[i],
nseasons_values[j], horizon, data[start_index_last_arg:end_index_last_arg])
      results[i, j, k] <- mse
    }
  }
}

# Вывод результатов
print(results)
```

Prophet:

```
library(readxl)
library(prophet)

# Загрузим данные
#data <- read_excel("data.xlsx", col_names = FALSE, range = "A2:A88")$...1
data <- read_excel("data.xlsx", col_names = FALSE, range = "B2:B249")$...1



# Функция для оценки точности модели Prophet
evaluate_prophet_model <- function(horizon, data2) {

  m <- prophet(df)
  # R
  future <- make_future_dataframe(m, periods = horizon)
  tail(future)

  forecast <- predict(m, future)
  tail(forecast[c('ds', 'yhat', 'yhat_lower', 'yhat_upper')])

  #print(forecast)

  #plot(m, forecast)

  # Создание имени файла
  filename <- paste0("plot_", interval_values[k],"_" ,data2, ".png")

  # Сохранение графика
```

```
  png(filename)

  # Создание графика
  plot(data2, type = "l", col = "blue", xlab = "Index", ylab = "Value", ylim =
c(15, 30), main = "Data2 vs Forecast")
  lines(forecast$yhat[1:horizon], col = "red")  # Добавление линии для прогноза

  legend("topright", legend = c("data2", "forecast"), col = c("blue", "red"),
lty = 1, cex = 0.8)  # Добавление легенды

  # Clear the Plot Window
  dev.off()


  # # Создание и обучение модели Prophet
  # df <- data.frame(ds = seq_along(data), y = data)
  # model <- prophet(df)
  # print(model)
  # # Создание фрейма для предсказаний
  # future <- make_future_dataframe(model, periods = horizon, include_history =
FALSE)

  # # Получение прогноза
  # forecast <- predict(model, future)
  #print(data2)
  #print(forecast$yhat[0:(horizon)])
  #print(length(data2))
  #print(length(forecast$yhat[0:(horizon)]))
  # # Вычисление MSE
  mse <- mean((data2 - forecast$yhat[1:horizon])^2)
  return(mse)
}

# Различные значения параметров
horizon <- 20
#interval_values <- c(20, 40, 60, 87 - horizon)  # Различные длины мерных
интервалов
interval_values <- c(60, 120, 180, 248 - horizon)  # Различные длины мерных
интервалов

# Создание матрицы для сохранения результатов
results <- array(NA, dim = c(length(interval_values)), dimnames =
list(interval_values))

# Оценка точности для разных значений параметров
for (k in seq_along(interval_values)) {

  dataframe <- read.csv("data2.csv")

  #print(dataframe)
  df <- head(dataframe, -horizon)
  #print(df)
  df <- tail(df, interval_values[k])
  #print(df)
  # Вычисление начального индекса для последнего аргумента функции
evaluate_bsts_model
  start_index_last_arg <- length(data) - horizon + 1
  # Вычисление конечного индекса для последнего аргумента функции
evaluate_bsts_model
```

```
  end_index_last_arg <- length(data)

  mse <- evaluate_prophet_model(horizon,
data[start_index_last_arg:end_index_last_arg])
  results[k] <- mse
}

# Вывод результатов
print(results)
```

SSA:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pySSA.core import MSSA
from pySSA.simple import SSA

N = 50 # Величина выборки
M = 9 # M значений вперед
r = 5 # r - количество компонент
L = 20 # L - ширина окна для построения траекторного пространства ряда

DATA_DIR = "./data/"
DATASET = DATA_DIR + 'data.xlsx'

data = pd.read_excel(DATASET)
time_series = data["smoothed_data"].tail(N)
time_series = time_series[::-1]

len_data = len(time_series)

ts = time_series[:(len_data - M)]
trust = time_series[(len_data - M):]

trust.reset_index(drop=True, inplace=True)
trust.index = trust.index + len_data - M

ts.reset_index(drop=True, inplace=True)



ssa = MSSA(ts)
ssa.embed(embedding_dimension=L)
ssa.decompose()
ssa.group_components(r)

C, f_L = ssa.L_reccurent_forecast(M)
print(f_L)
conf_int = ssa.conf_int()
print(conf_int)


# Вычисление квадратов разностей
squared_errors = (f_L.iloc[-M:,0] - trust) ** 2

# Вычисление среднего значения квадратов ошибок
mse = np.mean(squared_errors)
```

```
# Вычисление RMSE
rmse = np.sqrt(mse)

print("RMSE trust vs pred:", rmse)

# Вычисление квадратов разностей
squared_errors = (f_L.iloc[:-M,0] - ts) ** 2

# Вычисление среднего значения квадратов ошибок
mse = np.mean(squared_errors)

# Вычисление RMSE
rmse = np.sqrt(mse)

print("RMSE fact vs model:", rmse)

plt.figure(figsize=(12,6))
plt.title("Компоненты SSA", fontsize=20)
#plt.ylim(bottom=-0.035, top=0.025)
for i in range(r):
    plt.plot(C[:,i], label="Компонент %s"%i)
plt.legend()
plt.show()


trust_indices = np.arange(len(ts), len(ts) + len(trust))

plt.figure(figsize=(12,6))
plt.title("SSA прогноз", fontsize=20)
plt.plot(f_L.iloc[:-M,0], label="Модель")
plt.plot(ts, label="Выборка")
plt.plot(f_L.iloc[-M:,0], label="Прогноз")
plt.plot(trust, label="Истинные данные")
plt.legend()
plt.show()

# Вычисление остатков
residuals = trust - f_L.iloc[-M:,0]

# Построение графика остатков
plt.figure(figsize=(12,6))
plt.scatter(np.arange(len(residuals)), residuals)
plt.axhline(y=0, color='r', linestyle='-')  # линия нулевых остатков
plt.xlabel('Индекс')
plt.ylabel('Остатки')
plt.title('График остатков между истинными и предсказанными значениями')
plt.show()


# Вычисление остатков
residuals = ts - f_L.iloc[:-M,0]

# Построение графика остатков
plt.figure(figsize=(12,6))
plt.scatter(np.arange(len(residuals)), residuals)
plt.axhline(y=0, color='r', linestyle='-')  # линия нулевых остатков
plt.xlabel('Индекс')
plt.ylabel('Остатки')
plt.title('График остатков между фактическими данными и полученной моделью')
```

```
plt.show()
```