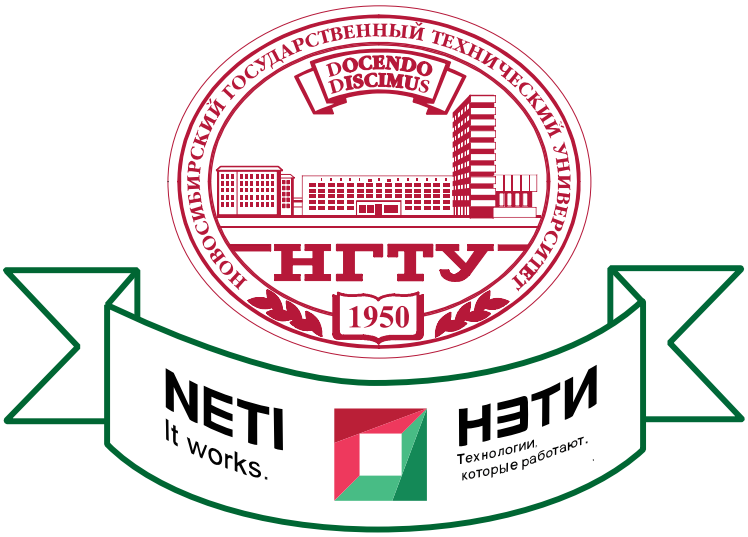
Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«Новосибирский государственный технический университет»

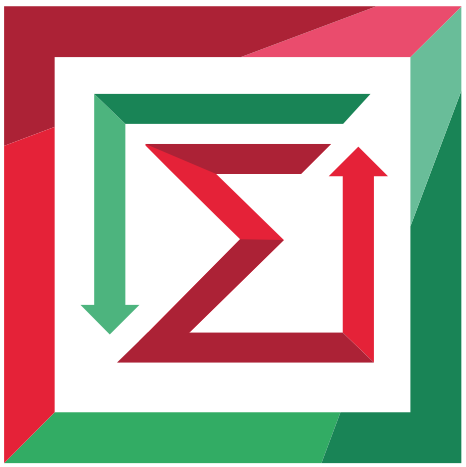


Теоретической и прикладной информатики

Лабораторная работа № 2

по дисциплине «Компьютерное моделирование»

Построение 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. Формулировка задания

Для каждого временного ряда x(t), y(t):

1. Построить и исследовать точность интегрированной модели авторегрессии скользящего среднего (ARIMA). Модель использует три основных параметра (p, d, q), которые выражаются целыми числами. Потому модель также записывается как ARIMA(p, d, q).

- p – порядок авторегрессии (AR), который позволяет добавить предыдущие значения временного ряда.
- d – порядок интегрирования (порядок разностей исходного временного ряда). Он добавляет в модель понятия разности временных рядов (определяет количество прошлых временных точек, которые нужно вычесть из текущего значения).

Для нестационарного временного ряда устанавливается параметр d=1, для стационарного d=0.

- q – порядок скользящего среднего, который позволяет установить погрешность модели как линейную комбинацию наблюдавшихся ранее значений ошибок.

Исследовать различную параметризацию модели ARIMA (параметры определить на основе критерия Акаике), установить оптимальные значения параметров для различных длин мерных интервалов. Результаты представить в виде таблицы:

Длина мерного интервала	p	q	Среднеквадратическое отклонение по уровню доверительной вероятности 0,95.

2. Построить и исследовать точность модели структурного временного ряда (BSTS). Исследовать различную параметризацию модели, установить оптимальные значения параметров для различных длин мерных интервалов. Результаты представить в виде таблицы из п.1.
3.

3.1. Выделить

- трендовую составляющую на основе полиномиальной регрессионной модели (можно использовать линейную, квадратичную, экспоненциальную функцию, линейную комбинацию многочленов (в т.ч. многочленов Чебышева));
- сезонную (гармоническую) составляющую на основе Фурье-анализа;
- остаточную составляющую (проверить имеют ли остатки нормальное распределение).

3.2. Сравнить полученные в п.3 результаты с аддитивной нелинейной регрессионной моделью (пакет Prophet).

Исследовать точность моделей из п.3.1 и 3.2 для различных длин мерных интервалов. Результаты представить в виде таблицы из п.1.

2. Описание выполненных действий

Построение ARIMA

Красным цветом лучшие по критерию Акаике.

Зеленым цветом лучшие по MSE.

Исследование на данных-1:

Длина мерного интервала	p	q	Среднеквадратическое отклонение по уровню доверительной вероятности 0,95.
20	1	3	0.000994
	1	1	0.000079
	1	4	0.000096
	4	1	0,004988
40	1	3	0.000765
	1	1	0.000090
	1	4	0,00018
	4	1	0,00051
60	3	3	0.000179
	1	1	0.000094
	1	4	0,000109
	4	1	0,000114
67	1	3	0.001018
	1	1	0.000103
	1	4	0,000173

	4	1	0,000119

Наиболее подходящими параметрами можно назвать p = 1, q = 1. Наиболее точна модель при длине мерного интервала = 67

Исследование на курсе валют:

Длина мерного интервала	p	q	Среднеквадратическое отклонение по уровню доверительной вероятности 0,95.
60	1	3	0.149321
	2	1	0.127097
	1	4	0,143038
	4	1	0,128038
120	1	3	0.147103
	2	1	0.126864
	1	4	0,142406
	4	1	0,128763
180	2	4	0.146498
	3	3	0.124830
	1	4	0,144325
	4	1	0,128981
228	4	3	0.159123
	3	3	0.123449
	1	4	0,144098

	4	1	0,128962

Наиболее подходящими параметрами можно назвать p = 3, q = 3. Наиболее точна модель при длине мерного интервала = 228.

Заметим, что для курса валют значения MSE больше, так как данные обладают более сложной структурой и менее предсказуемы. Модель не обладает тенденцией уточнятся с ростом длин мерных измерений.

Построение BSTS

Исследование на данных-1:

Длина мерного интервала	Значения niter	Значения nseasons	Среднеквадратическое отклонение по уровню доверительной вероятности 0,95.
20	500	6	0.002839374
	1000	6	0.004966074
	1500	12	0.004508740
	1500	24	0.005544352
40	500	6	0.001698797
	1000	6	0.002134354
	1500	12	0.0002027616
	1500	24	0.0014294351
60	500	6	0.001009533
	1000	6	0.001042137

	1500	12	0.0001952372
	1500	24	0.004714057
67	500	6	0.001210375
	1000	6	0.001426513
	1500	12	0.0001363947
	1500	24	0.006672933

Видим, что наиболее подходящие параметры niter = 1500, nseasons = 12. Модель не уточняется с ростом длины мерного интервала.

Исследование на курсе валют:

Длина мерного интервала	Значения niter	Значения nseasons	Среднеквадратическое отклонение по уровню доверительной вероятности 0,95.
60	500	6	1.893073
	1000	6	1.861183
	1500	12	1.3097368
	1500	24	5.021768
120	500	6	1.365517
	1000	6	1.378934
	1500	12	0.8836305
	1500	24	4.117229
180	500	6	1.877787
	1000	6	2.025478

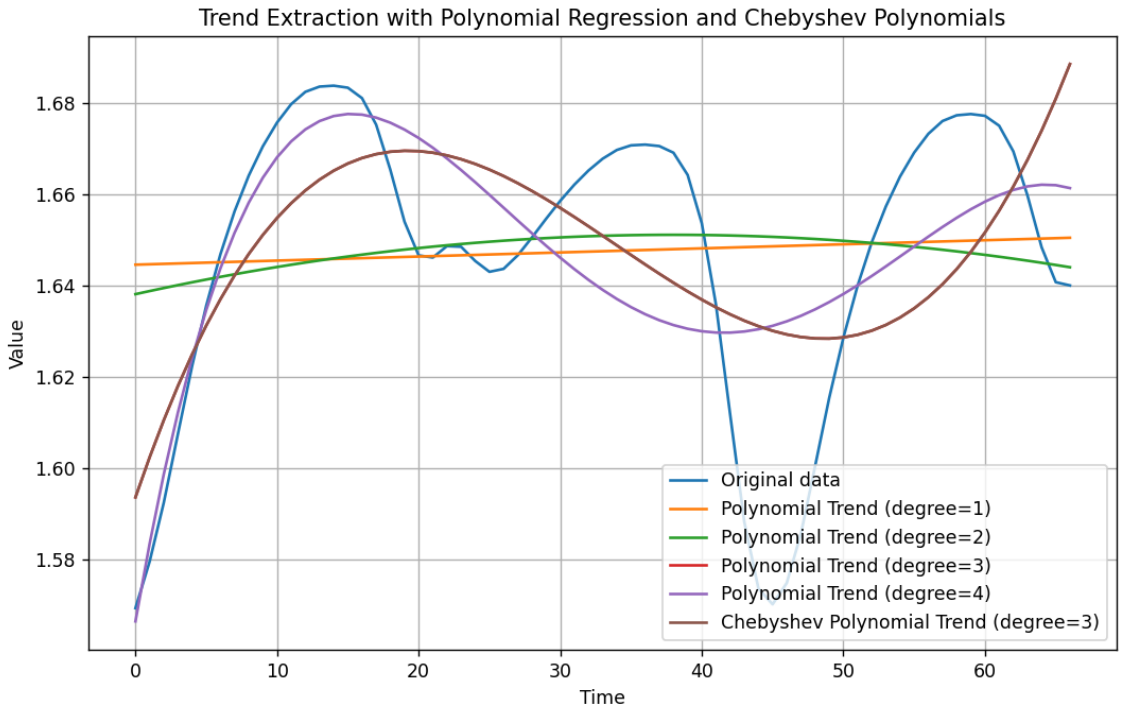
	1500	12	0.8597027
	1500	24	3.401474
228	500	6	1.778447
	1000	6	2.020571
	1500	12	0.8613856
	1500	24	2.782722

Видим, что наиболее подходящие параметры niter = 1500, nseasons = 12. Модель не уточняется с ростом длины мерного интервала.

Можем наблюдать, что для курса валют значения MSE больше, так как данные обладают более сложной структурой и менее предсказуемы.

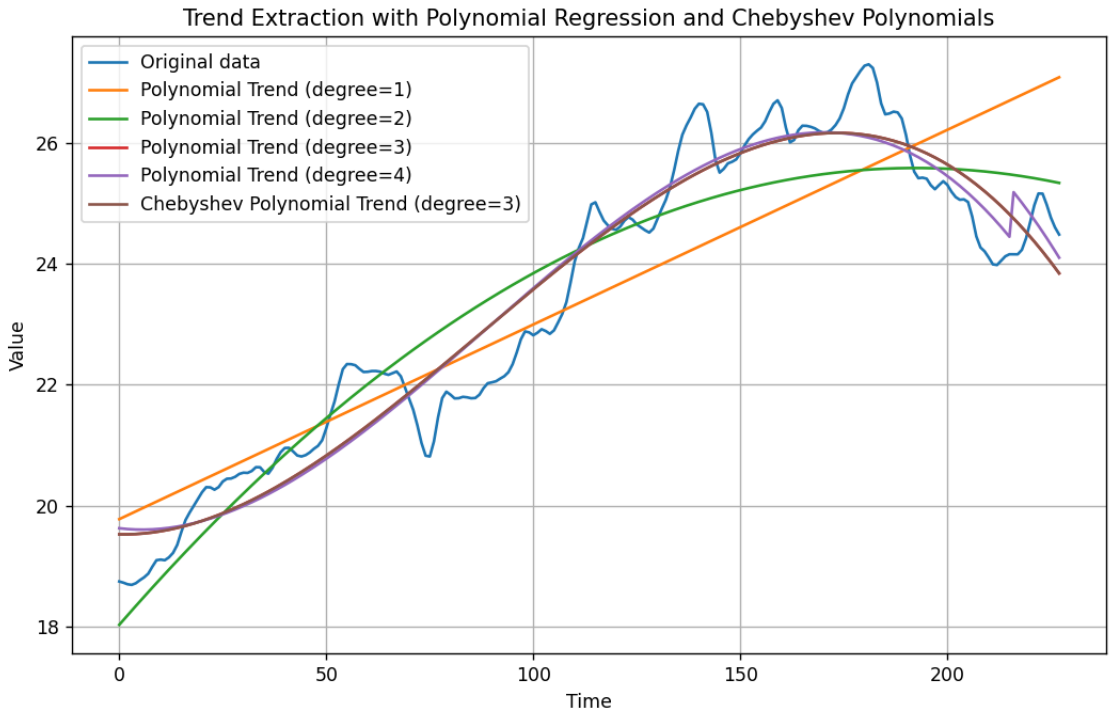
Выделение тренда

Исследование на данных-1:



Наиболее подходящий тренд: Чебышева 30 степени.

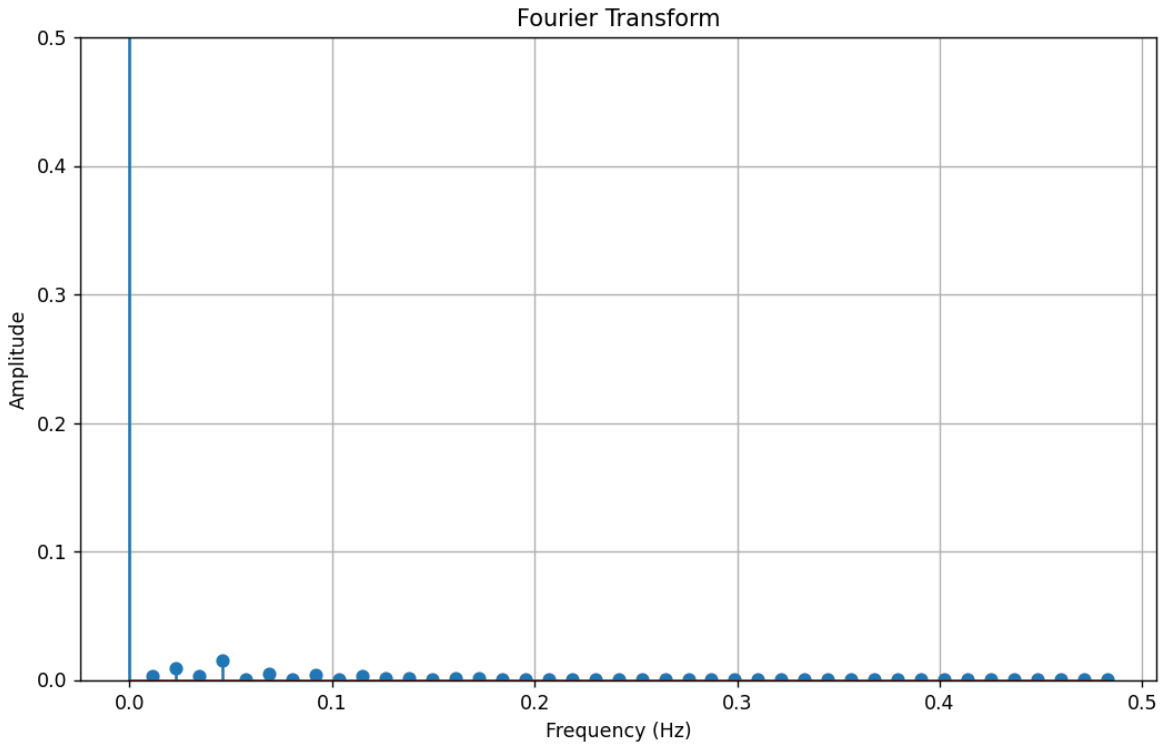
Исследование на курсе валют:

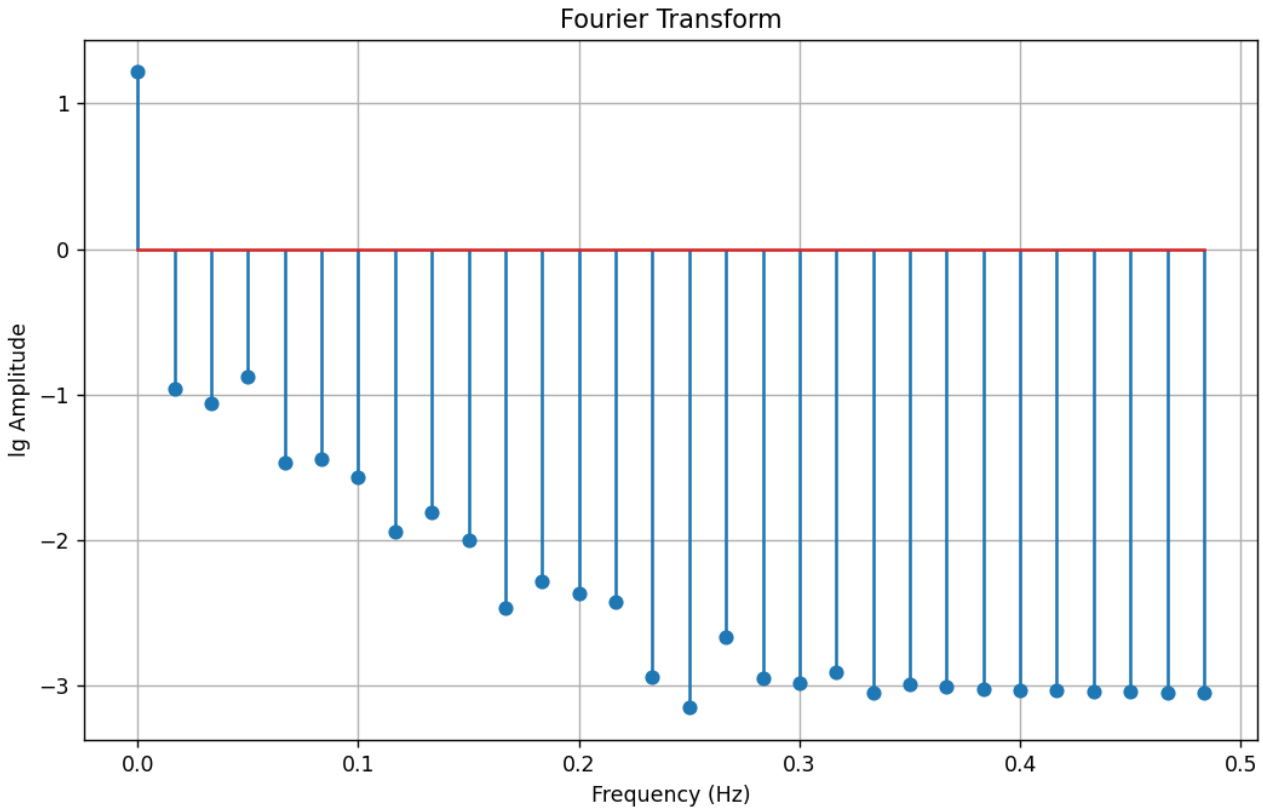


Наиболее подходящий тренд: Чебышева 30 степени.

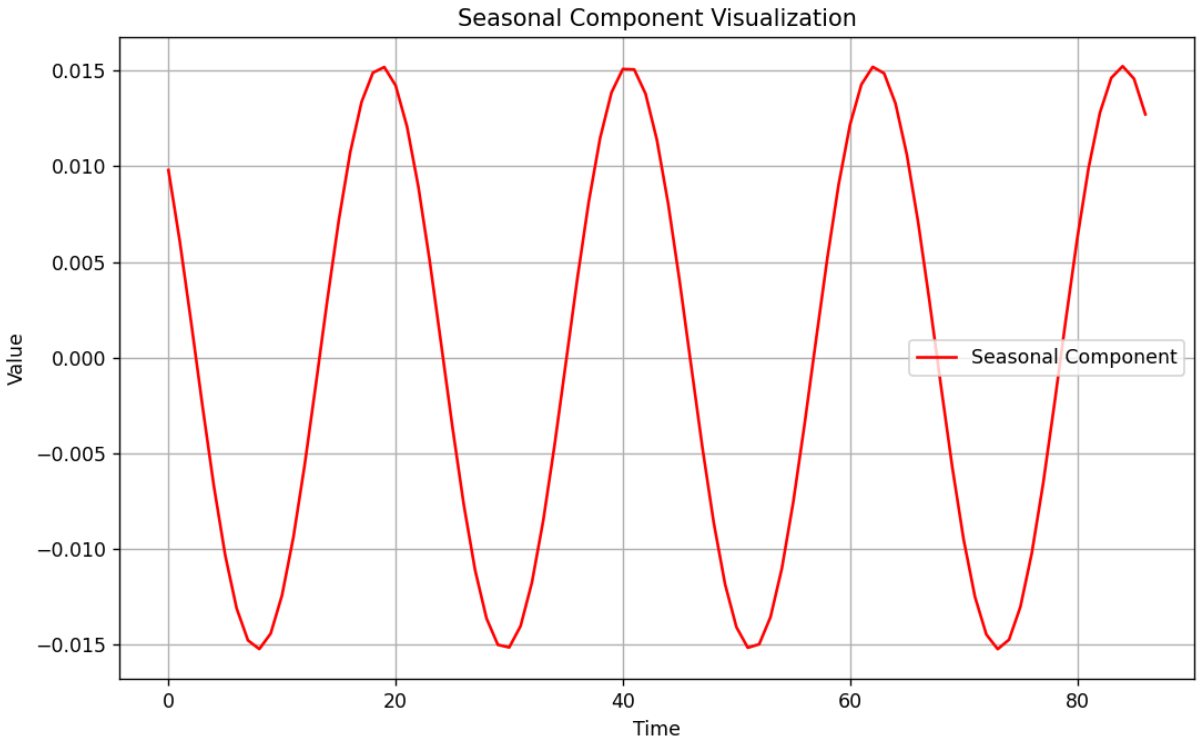
Выделение сезонной составляющей

Исследование на данных-1:

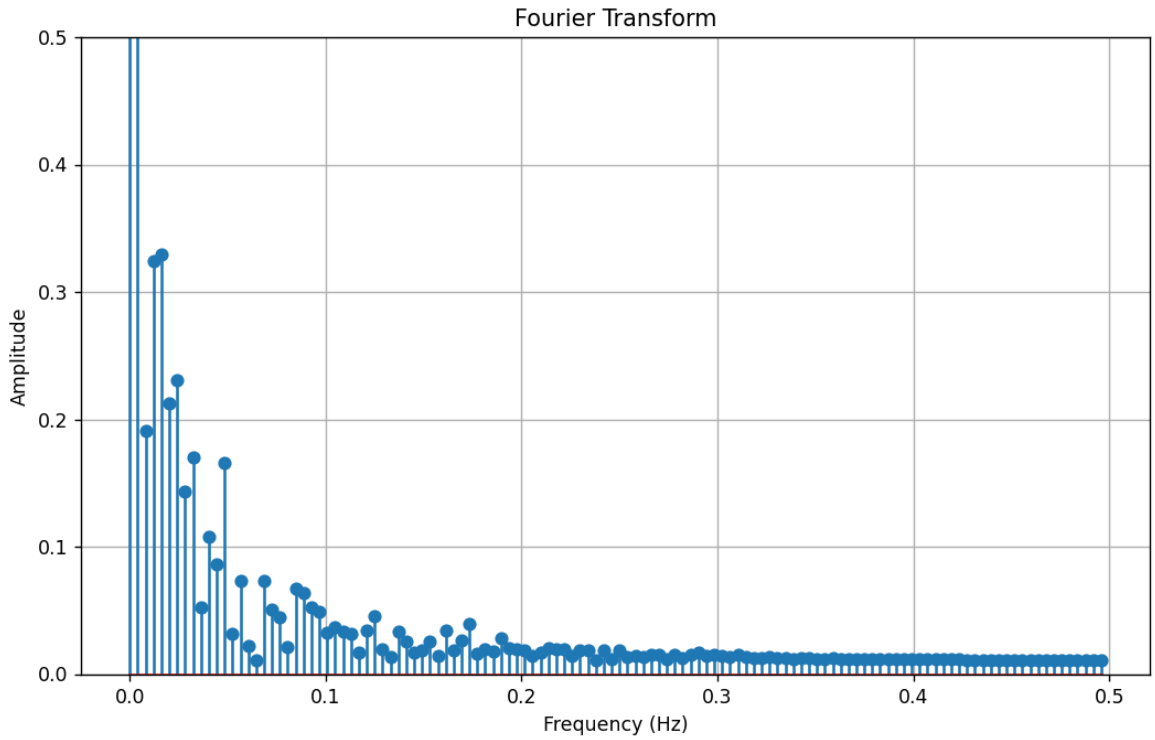


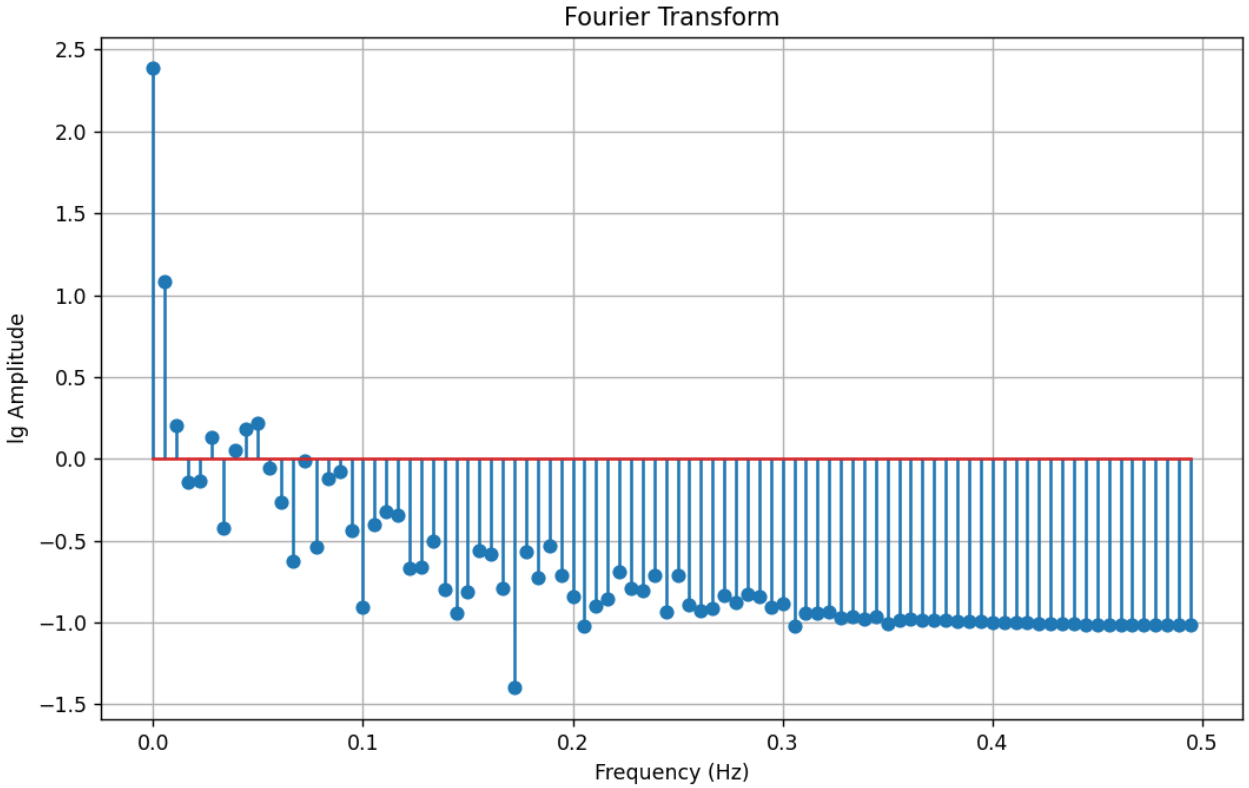


Наблюдаем очень низкую амплитуду на всех частотах.

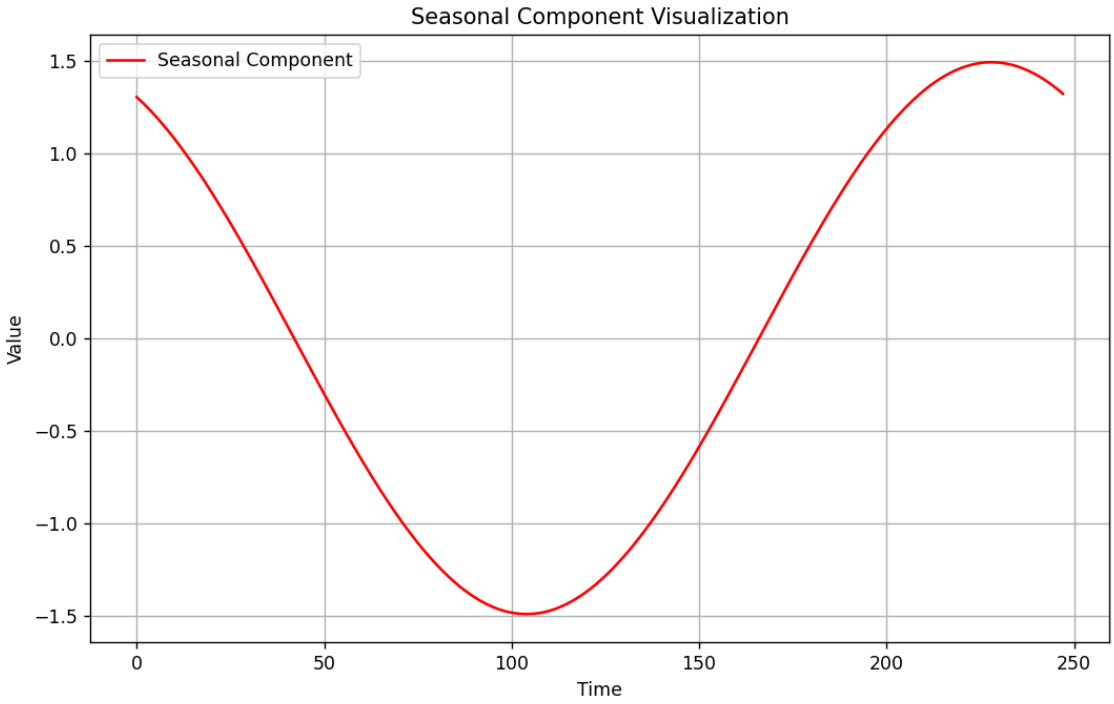


Исследование на курсе валют:



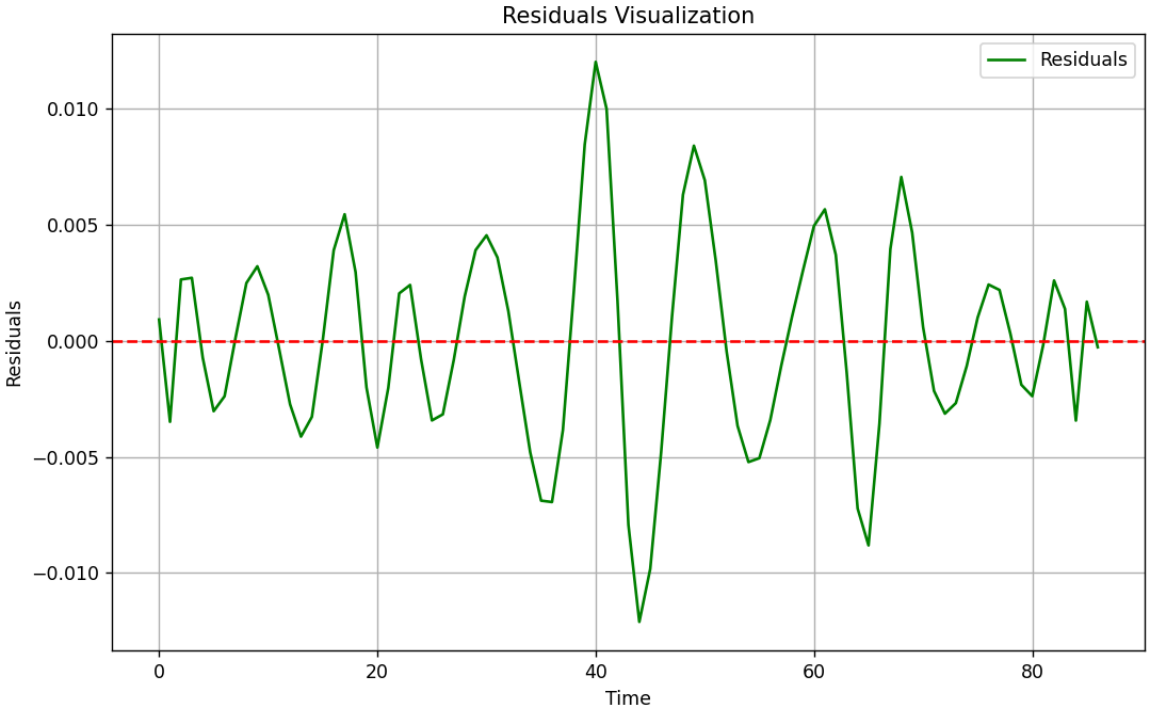


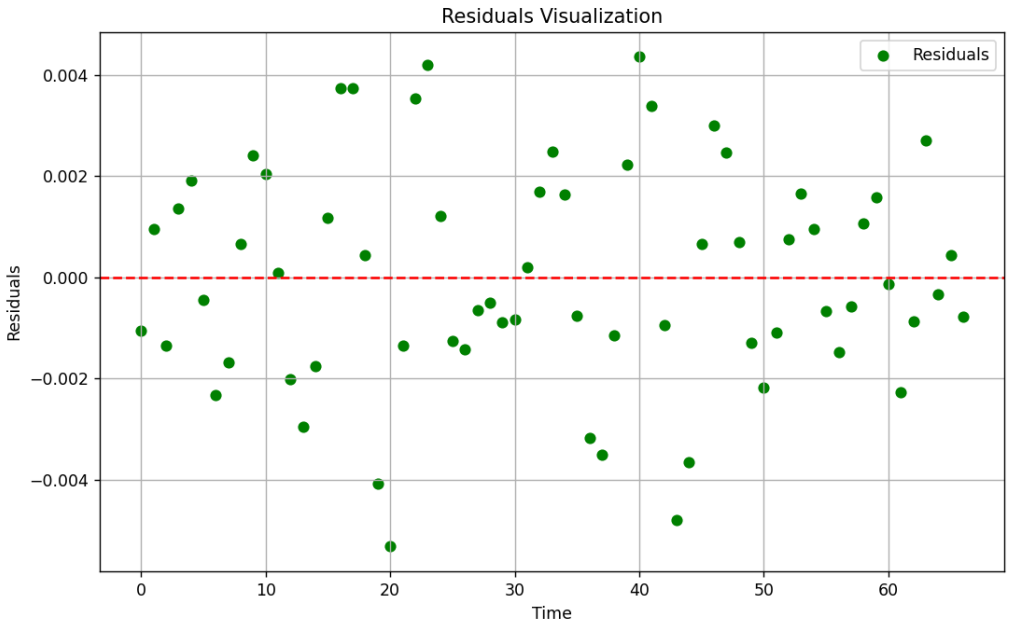
Наблюдаем снижение амплитуды с ростом частоты.



Выделение остаточной составляющей

Исследование на данных-1. Пример нормального остатка:



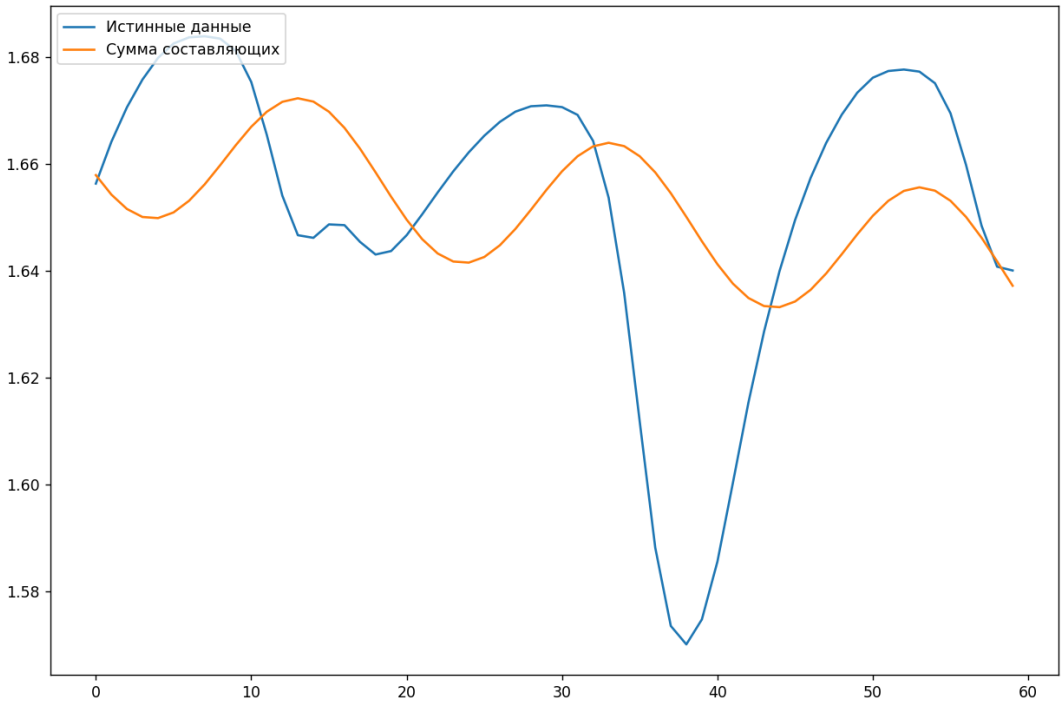


Остатки имеют нормальное распределение.

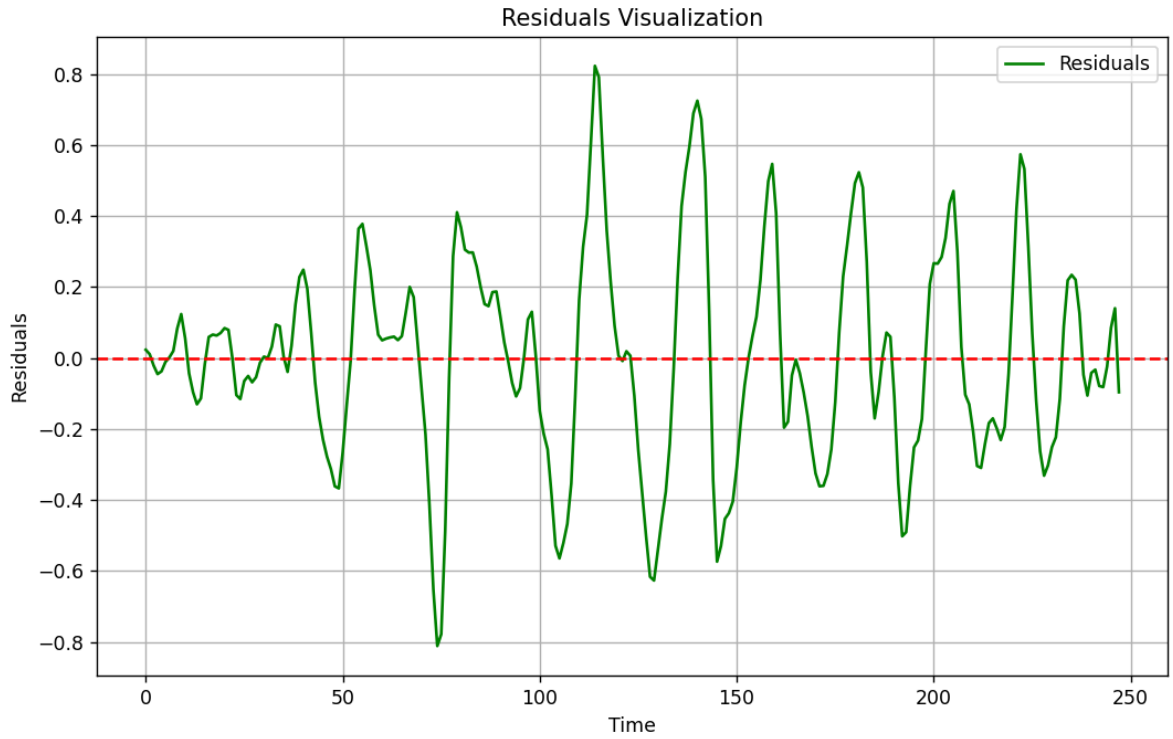
Длина мерного интервала	Тренд	Среднеквадратическое отклонение по уровню доверительной вероятности 0,95.
20	1 степень	0.003094100418160484
	2 степень	0.07955268012764653
	3 степень	0.07169087151764933
	4 степень	0.007520268587727646
40	1 степень	3.722518647398664e-05
	2 степень	0.010593501372563756
	3 степень	0.00025261495066857226
	4 степень	2.7292195944485953
60	1 степень	0.00025309727048372166
	2 степень	0.0017603481849853039

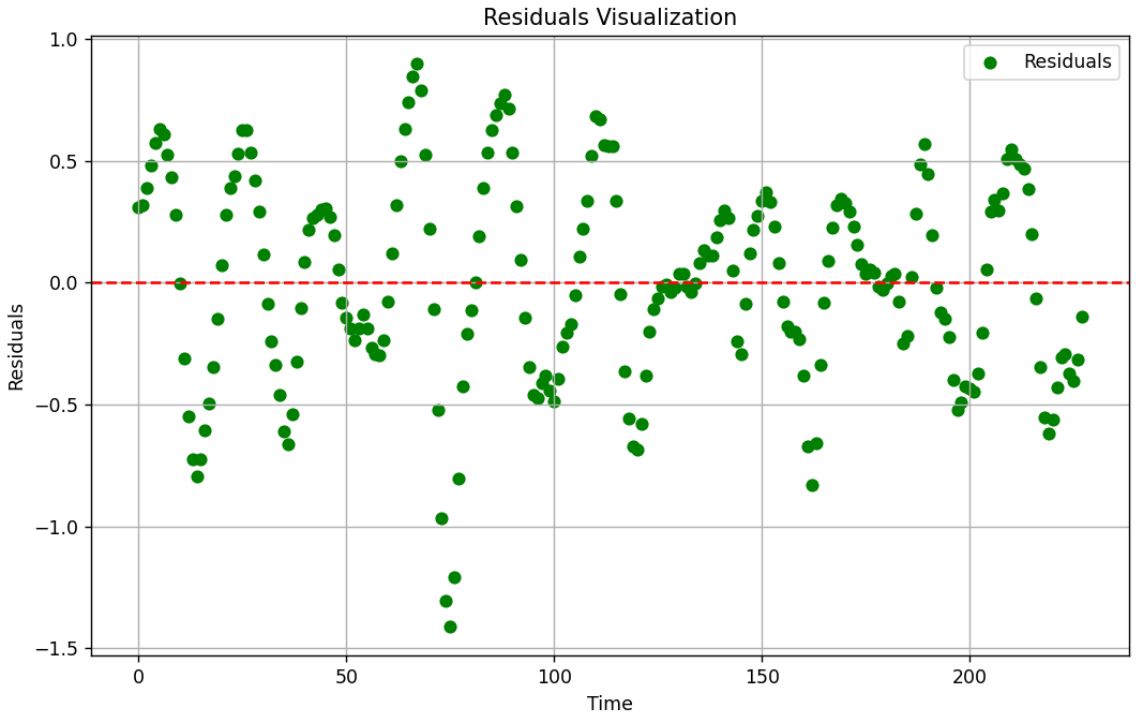
	3 степень	0.0143302429441624 08
	4 степень	0.0035892304442191533
67	1 степень	0.0003758320941348133
	2 степень	0.0006719693294759311
	3 степень	0.03734165720538154
	4 степень	0.011371686823214561

Легко заметить, что наименьшая ошибка при линейной функции.



Исследование на курсе валют. Пример нормального остатка:

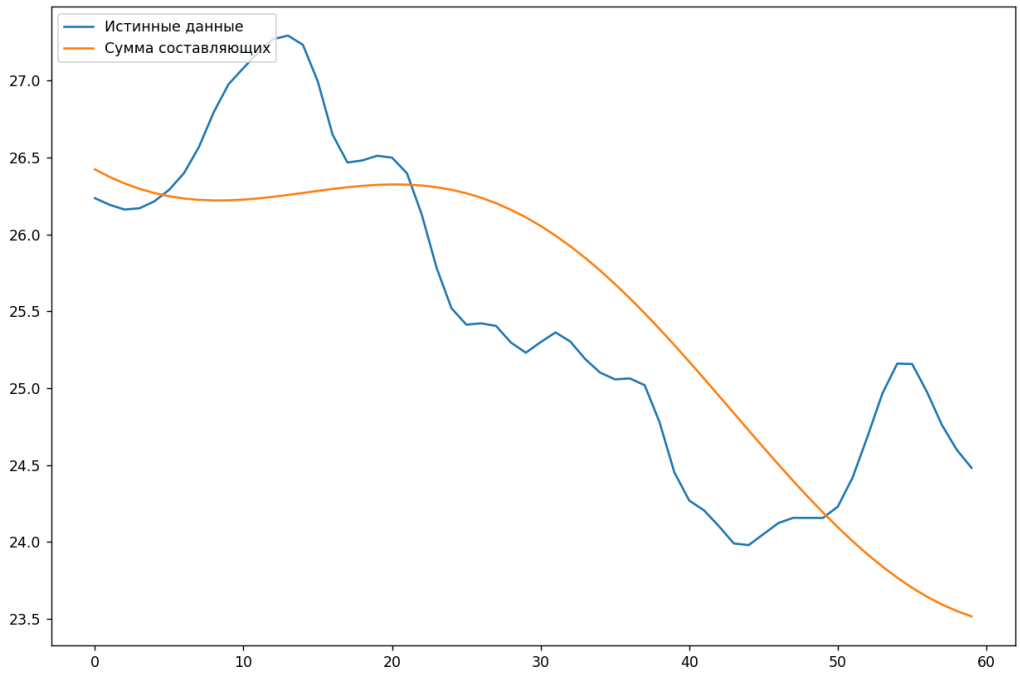




Остатки имеют нормальное распределение.

Длина мерного интервала	Тренд	Среднеквадратическое отклонение по уровню доверительной вероятности 0,95.

60	1 степень	0.21537485887890026
	2 степень	0.04971328790961547
	3 степень	33.04207404528106
	4 степень	5.288204451761135
120	1 степень	0.30014124140000256
	2 степень	5.537344862479001
	3 степень	3.520979191331218
	4 степень	2.547850279875055
180	1 степень	5.707904589306779
	2 степень	0.2604266964146235
	3 степень	11.171105141131259
	4 степень	0.5681376860868153
228	1 степень	12.664185160728 41
	2 степень	1.7214085501292415
	3 степень	1.8953633981775249
	4 степень	8.586617210444652



Чаще всего наименьшая ошибка при квадратичной функции.

В общем итоге отметим, что точность снижается с ростом длины мерного интервала.

Построение аддитивной нелинейной регрессионной модели

Исследование на данных-1:

Длина мерного интервала	Среднеквадратическое отклонение по уровню доверительной вероятности 0,95.
20	0.0004028378
40	0.0014800858
60	0.0005995639
67	0.0007764467

Можно сделать вывод, что не наблюдается тенденции по снижению или увеличению среднеквадратического отклонения с ростом длины мерного интервала.

Исследование на курсе валют:

Длина мерного интервала	Среднеквадратическое отклонение по уровню доверительной вероятности 0,95.
60	4.9736846
120	0.4052885
180	7.2193090
226	27.8817523

Напрашивается вывод о том, что наблюдается тенденция по увеличению среднеквадратического отклонения с ростом длины мерного интервала. Результат достаточно плох для курса валют, но с длиной мерного интервала 120 результат хороший.

3. Вывод

Подведем итог в виде сравнения всех построенных моделей. Наиболее хорошо для обеих выборок себя проявила модель ARIMA и модель, построенная при помощи тренда и сезонной составляющей (при верно подобранном тренде). Если оценивать модели отдельно для каждой выборки, то для данных-1 так же хорошо подошла аддитивная нелинейная регрессионная модель. Аддитивная нелинейная регрессионная модель оценила курс валют достаточно точно при длине мерного интервала = 120, она имеет потенциал стать наиболее подходящей, как и модель ARIMA.

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)
```