МИНИСТЕРСТВО НАУКИ И ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Институт

ИЯТШ

ЛАБОРАТОРНАЯ РАБОТА № 2

ЭКОНОМЕТРИЧЕСКИЕ МОДЕЛИ

по дисциплине:

**Статистическое моделирование и прогнозирование**

Вариант № 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Выполнил:** |  | | | | |
| студент группы 0В21 |  |  | Дзебан А.А. |  | 11.06.2025 |
|  |  |  |  |  | Дата сдачи |
| **Проверил:** | Рекундаль О.И. | | | | |
| доцент |  |  |  |  |  |
|  |  |  |  |  |  |

Томск – 2025

**Теоретическая часть:**

Выборочной автоковариацией k-го порядка временного ряда называется число

.

Выборочной автокорреляцией k-го порядка временного ряда называется число .

Статистика Дарбина-Уотсона определяется так:

Выдвигаем нулевую гипотезу

: , где - первая автокорреляция (остальные автокорреляции статистически не проверяются).

Нулевая гипотеза о нулевой автокорреляции подтверждается, если

.

отклоняется в пользу альтернативы о наличии положительной автокорреляции, если DW. отклоняется в пользу альтернативы о наличии отрицательной автокорреляции, если .

Для выявления автокорреляции проводится тест Льюинга-Бокса:

Проверяется

H0: отсчеты временного ряда статистически независимы

H1: отсчеты временного ряда не являюются независимыми.

Статистика:

Если , гипотеза H0 принимается.

**АВТОРЕГРЕССИОННАЯ МОДЕЛЬ *AR*(*p*)**

Рассмотрим математическую модель, которая позволяет обработать эмпирические данные (например, котировки акций и т.п.) и оценить их будущее значение. Предлагаемая авторегрессионная модель позволяет получить оценку временного ряда ***hn***, используя знания о его предыдущих состояниях .

Рассмотрим авторегрессионную модель *AR*(*p*) порядка *p*:

где *a*0, *a*1, ..., *ap*, – некоторые коэффициенты, – нормальное распределенные случайные величины с нулевым средним и дисперсией единица.

В общем случае оценку коэффициентов модели *AR(p)* ленче всего проводить с помощью рекуррентных формул Юла-Уокера:

где – выборочные автокорреляции процесса - неизвестные коэффициенты модели *AR(p)*, *j=1, 2, …, p*. При этом кулевой коэффициент выражается из формулы для математического ожидания:

Для проверки статистической гипотезы о нестационарности ряда значений модели используется критерий Дики-Фулера (Dickey-Fuller test, ADF). Нулевая гипотеза *H0* состоит в том, что ряд нестационарен и = 1 при альтернативной гипотезе, что ряд стационарен и :

Критическая статистика: , где , где – диагональный элемент обратной матрицы , *Х –* центрированные столбцы данных.

Нулевая гипотеза *H*0 принимается, если *t*кр<γ. В противном случае принимается альтернативная гипотеза (при γ< *t*кр).

Для выбора параметра p существует несколько возможных способов:

1. Принять p, соответствующее максимальному статистически значимому порядку автокорреляции
2. Необходимо выбирать модель, у которой значение AIC меньше.

**АВТОРЕГРЕССИОННАЯ МОДЕЛЬ СО СКОЛЬЗЯЩИМ СРЕДНИМ *ARMA*(*p,q*)**

Обобщим модель AR(p), добавив к ней нормально распределенные ошибки наблюдений , с нулевым средним и дисперсиями , которые вносят так называемую скользящую среднюю ошибку, и убрав константу:

По аналогии с *AR(p)* – процессом, *ARMA(p,q)* будет стационарным, если первые m его автокорреляций остатков модели равны нулю. Для проверки гипотезы используют статистику Бокса – Пирса:

где n – число данных, p, q – параметры модели, – выборочные автокорреляции между эмпирическими и теоретическими значениями модели:

Для этой статистики нулевая гипотеза о равенстве нулю первых m автокорреляций остатков подтверждается, если при заданном уровне значимости .

Для оценки параметров а необходимо решить систему:

…

Для нахождения решения системы относительно неизвестных вместо теоретических значений автоковариации будем использовать их выборочные аналоги:

Найдя оценки , перейдем к процессу *,* для которого рассчитаем первые q выборочные автокорреляции. Их используем для вычисления оценок параметров . Действительно, в соответствие обобщенной модели фактически

– процесс скользящего среднего. Его автокорреляции можно сравнить с выборочными аналогами:

где – выборочные автокорреляции.

**Модель ARIMA(p,q)**

Модель ARIMA – обобщение модели ARMA на случай нестационарности исходного ряда, но стационарности дифференцированного ряда порядка d.

То есть для ряда вида:

**GARCH(p,q)**

Модель GARCH позволяет спрогнозировать волатильность временного ряда, используя формулу:

Причем:

Где

– относительные приращения цен,

Если

Если же

Пример: пусть *p*=1 и *q*=1. Тогда модель будет иметь вид:

где – коэффициенты модели, подлежащие оценке – долговременное среднее отклонение в структуре данных, , – относительные приращения значений временного ряда – волатильность, . нужно выбирать таким образом, чтобы максимизировать функцию правдоподобия:

,

где

**Практическая часть:**

**Задание 1:**

В соответствии с вариантом задания построить модель ***AR*(*p*) с оптимальным значением лага**, для чего

1. Вычислить выборочные автокорреляции до порядка *n*=10 включительно;
2. проверить исторические данные на случайность, вычислив статистику Льюнга– Бокса;
3. оценить коэффициенты модели *AR*(*p*) с помощью формул Юла – Уокера;
4. подобрать оптимальный параметр *p*, проверяя при различных лагах критерий единичного корня Дики-Фулера;
5. подобрать параметр *p* с помощью информационного критерия Акаике. Сравнить его с найденным в п.4 лагом *p*;
6. построить алгоритмом *AR*(*p*) прогноз на 25 день торгов.

Выборочные автокорреляции вычислим по формуле (1.1.):

|  |  |
| --- | --- |
| t |  |
| 0 | 1 |
| 1 | 0.7233 |
| 2 | 0.359 |
| 3 | 0.0936 |
| 4 | -0.117 |
| 5 | -0.4368 |
| 6 | -0.5991 |
| 7 | -0.521 |
| 8 | -0.38 |
| 9 | -0.2632 |
| 10 | -0.0546 |

Q-статистика Льюнга-Бокса по формуле (1.2.):

Нулевая гипотеза отвергается – ряд не случаен.

Построим авторегрессионную модель вида: . Для оценки коэффициентов регрессии воспользуемся рекуррентной формулой Юла-Уокера (1.3.):

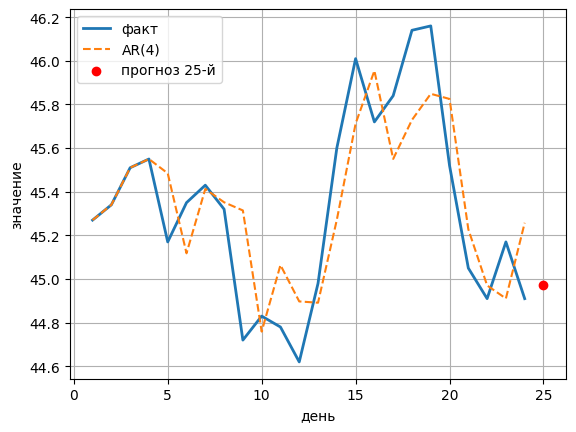
[a0 = 12.54154376, a1=0.72332287]

Подберём оптимальный параметр p, проверяя при различных лагах критерий единичного корня Дики-Фулера и информационный критерий Акаике. Воспользуемся статистикой DF (1.4.) и AIC(1.5) при различных значениях p и найдем критические статистики при

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| p | AIC | ADF | p-value | P\_crit |
| 1 | -2.341 | -2.249 | 0.189 | -3.005 |
| 2 | -1.673 | -1.578 | 0.495 | -3.0013 |
| 3 | -1.127 | -1.922 | 0.322 | -3.022 |
| 4 | -0.742 | -3.704 | 0.004 | -3.031 |
| 5 | -0.954 | -2.282 | 0.178 | -3.042 |

Заметим, что при p = 4 p-value < 0.05, то есть ряд является стационарным.

Построим на алгоритм на 25 день торгов:



Прогноз y₍₂₅₎ = 44.972

**Задание 2:**

В соответствии с вариантом задания построить модель ***ARMA*(*p*,1) с оптимальным значением лага *p***, для чего

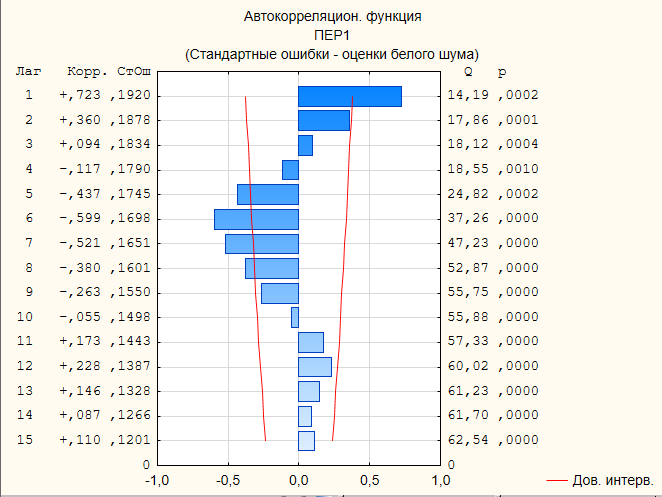
1. Вычислить выборочные автоковариации и автокорреляции до порядка *n*=10 включительно (они нужны для оценки параметров модели ***ARMA*(*p*,1)**). Построить по найденным автокорреляциям коррелограмму и оценить по ней значение лага *p* модели *ARMA*(*p*,1), если это возможно;
2. центрировав исходные данные и перейдя к вспомогательному ряду , оценить коэффициенты модели *ARMA*(*p*,1);
3. проверяя значимость статистики Бокса – Пирса, найти максимальное значение *m* равных нулю первых автокорреляций ошибок центрированной модели *ARMA*(*p*,1). Будет ли такой *ARMA*(*p*,1) – процесс стационарным?
4. подобрать параметр *p* с помощью информационного критерия Акаике. Сравнить его с найденным в п.1 лагом *p*;
5. возвращаясь к исходным данным , построить методом *ARMA*(*p*,1**)** прогноз на 25 день торгов;
6. если построить модель *ARMA*(*p*,1**)** невозможно, продифференцируйте исходный временной ряд и постройте модель *ARIMA*(*p*,1**)**. Возвращаясь к исходным данным, постройте методом *ARIMA*(*p*,1**)** прогноз на 25 день торгов;
7. с помощью программного пакета Statistica проверить качество сделанного прогноза на 25 день торгов, построив *ARMA*(*p*,1) для исходных данных и перейдя по необходимости к дифференцированному с первым порядком ряду (если построение требуемой модели напрямую невозможно).

Автокорреляция: [1, 0.7233228695750802, 0.35960085654492824, 0.09369285909262455, -0.11739352082095013, -0.4368386923480597, -0.5991180902737538, -0.5210551195136935, -0.3802301652887395, -0.26329922484678076, -0.05461377576521095]

Автоковариация: [0.18571597222222255, 0.13433260995370389, 0.06678362268518533, 0.01740026041666647, -0.021801851851852478, -0.08112792245370427, -0.11126579861111148, -0.09676825810185197, -0.07061481481481463, -0.048898871527777464, -0.010142650462962609]

Построим коррелограмму:

Поскольку r1>0.5 построение модели ARMA (p,1) невозможно. Будем строить модель ARIMA(p,1).



Продифференцируем ряд и получим следующие значения:

array([ 0.18954545, 0.05954545, -0.36045455, 0.19954545, 0.09954545,

-0.09045455, -0.58045455, 0.12954545, -0.03045455, -0.14045455,

0.37954545, 0.63954545, 0.42954545, -0.27045455, 0.13954545,

0.31954545, 0.03954545, -0.62045455, -0.45045455, -0.12045455,

0.27954545, -0.24045455])

Соответствующие автокорреляции:

[1,

0.21204691697558437,

-0.24230363774837796,

-0.0704480624231858,

0.28439945504374164,

-0.18243668209085978,

-0.46328641623659916,

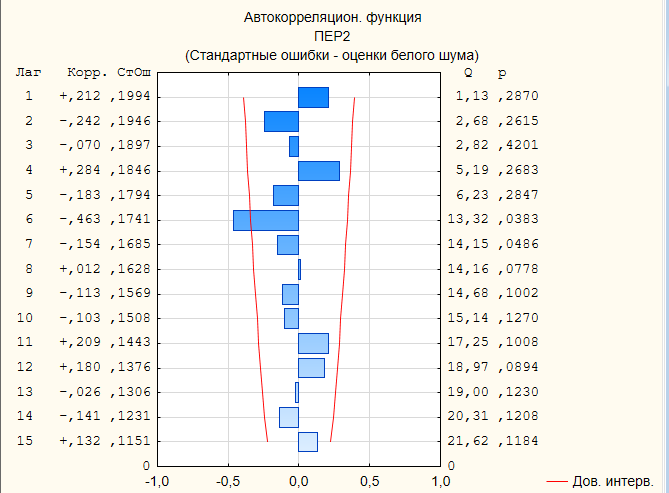
-0.15426534397683392,

0.012023092857543924,

-0.11293277073765678,

-0.10297997738253345]

Заметно, что следовательно, параметр принимает действительные значения и модель может быть построена.



Исходя из графиков автокорреляционных функций значение p составляет 6.

Оценим коэффициенты модели ARIMA(6,1) для дифференцированного центрированного ряда:

array([-0.05916667, 0.01083333, 0.18083333, 0.22083333, -0.15916667,

0.02083333, 0.10083333, -0.00916667, -0.60916667, -0.49916667,

-0.54916667, -0.70916667, -0.34916667, 0.27083333, 0.68083333,

0.39083333, 0.51083333, 0.81083333, 0.83083333, 0.19083333,

-0.27916667, -0.41916667, -0.15916667, -0.41916667])

Итоговая модель имеет следующий вид:

ARIMA(6,1,1) =

Проверяя значимость статистики Бокса-Пирса составим таблицу лагов и p-val:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Лаг | 1 | 2 | 3 | 4 | 5 | 6 |
| P-Val | 0.045 | 0.135 | 0.26 | 0.391 | 0.529 | 0.663 |

Итого значение m = 1;

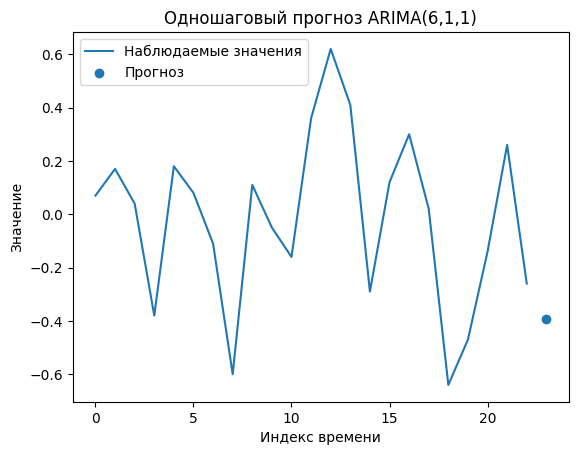
Исследуемый процесс не является стационарным.

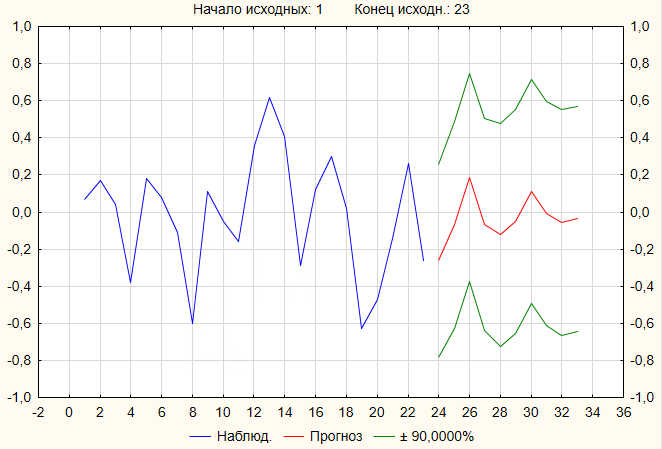
Подберем параметр p используя информационный критерий Акаике. Определим наименьшее значение AIC из итоговой таблицы:

|  |  |
| --- | --- |
| Лаг | AIC |
| 1 | -53.315 |
| 2 | -51.38 |
| 3 | -50.89 |
| 4 | -49.2 |
| 5 | -51.1 |
| 6 | -58.1 |
| 7 | -56.17 |

Таким образом наименьшее значение AIC соответствует p = 6, совпадает с результатами коррелограммы.

Построим предсказание для T = 25:





Заметно, что прогнозы имеют отклонения.

Задание 3:

В соответствии с вариантом задания построить модель ***GARCH*(1,1)**, для чего:

1. перейти к **относительным приращениям** исходных данных;
2. вычислить выборочное математическое ожидание и смещенную оценку дисперсии для последних *k*=5, 10, 15 значений временного ряда. Правда ли, что выборочное среднее очень близко к нулю?
3. Учитывая условие устойчивости ***GARCH*(1,1)**–процесса, оценить оставшиеся неизвестные коэффициенты, перебирая α и β, находя максимум функции правдоподобия с плотностями нормального распределения по вычисленным с помощью GARCH-модели дисперсиям .
4. вычислить отдельно две последовательности автокорреляций до порядка 5 включительно для рядов  (до применения GARCH) и  (после применения GARCH);
5. проверить значимость статистики Льюнга–Бокса для вычисленных в п.4. двух последовательностей автокорреляций. Показать, что для ряда статистика Льюнга-Бокса может быть значима (что означает стационарность временного ряда );
6. возвращаясь к исходным данным, построить методом ***GARCH*(1,1)** прогноз на 25 день торгов, если это возможно;
7. с помощью программного пакета Matlab проверить качество сделанного прогноза на 25 день торгов, построив ***GARCH*(1,1)** для исходных данных (использовать model = garch('garchlags',1,'archlags',1); [estM1,H,logL] = estimate(model,a), где a – ряд ценовых приращений; функцию forecast).

Переходя к относительным приращениям получим следующий ряд:

array([ 0.00154628, 0.00374945, 0.00087893, -0.00834248, 0.00398495,

0.00176406, -0.00242131, -0.01323919, 0.00245975, -0.00111532,

-0.00357302, 0.00806813, 0.0137839 , 0.00899123, -0.00630298,

0.00262467, 0.0065445 , 0.00043346, -0.01386482, -0.01032513,

-0.00310766, 0.00578936, -0.00575603])

Определим выборочное мат ожидание и смещенную оценку дисперсии для k 5,10,15

|  |  |  |
| --- | --- | --- |
| K | Мат ожидание | Дисперсия |
| 5 | -0.0545 | 4.52e-05 |
| 10 | -0.00149 | 5.23e-05 |
| 15 | 0.00031 | 5.46e-05 |

Таким образом действительно можно сделать вывод о близости к нулю выборочного среднего.

Оценим коэффициенты модели. Параметр V можно оценить по формуле (17). Тогда останется всего 2 неизвестных коэффициента α и β, которые будем перебирать с некоторым шагом α = i/100,β = j /100, 0 < α + β <1, i, j =1,99 и вычислять функции правдоподобия L.

де функция плотности распределения наблюдений.

Итоговая модель выглядит так:

Построим автокорреляции рядов , .

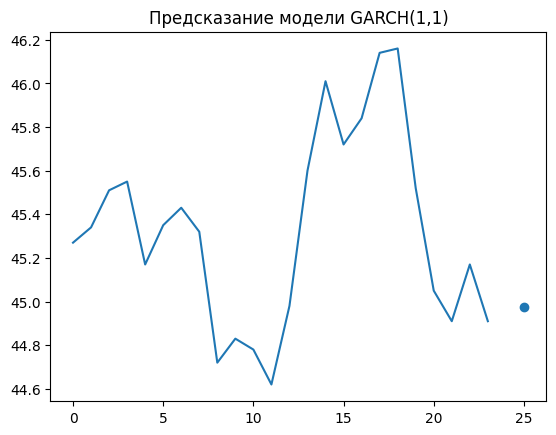
|  |  |  |
| --- | --- | --- |
| Лаг | Autocorr( | Autocorr( |
| 1 | 0.057 | 0.07 |
| 2 | -0.241 | -0.25 |
| 3 | -0.298 | -0.28 |
| 4 | -0.0029 | -0.04 |
| 5 | 0.2049 | 0.179 |

Определим статистику Льюнга-Пирса и подсчитаем p-value

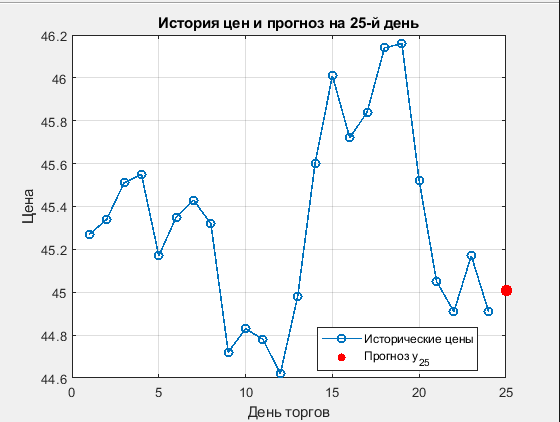
|  |  |  |
| --- | --- | --- |
| Ряд | Q | p-value |
| U | 5.5 | 0,35 |
|  | 8.2 | 0.042 |

Таким образом ряд можно считать стационарным.   
  
Проведем предсказание модели

Прогноз цены y\_(T+1) = 44.97416759294688



Вывод MATLAB:



Предсказания соответствуют ожиданиям.

**Вывод:**

**Листинг кода:**

**MATLAB**

**% 1) Задаём ваш ряд приращений a = diff(prices);**

**a = diff(prices);**

**% 2) Оцениваем GARCH(1,1) на a:**

**model = garch('GARCHLags',1,'ARCHLags',1);**

**[estM1,~,~] = estimate(model, a, 'Display','off'); % mean=0**

**% 3) Прогноз дисперсии одного шага вперёд:**

**HFor = forecast(estM1, 1, 'Y0', a); % HFor — это σ²\_{T+1}**

**% 4) Прогноз цены на следующий день:**

**price25\_hat = prices(end) + HFor;**

**fprintf('Прогноз дисперсии σ²\_{T+1} = %.6f\n', HFor);**

**fprintf('Прогноз цены y\_{T+1} = %.4f\n', price25\_hat);**

**Прогноз дисперсии σ²\_{T+1} = 0.098170**

**Прогноз цены y\_{T+1} = 45.0082**

**>> % 1) Задаём ваш ряд приращений a = diff(prices);**

**a = diff(prices);**

**% 2) Оцениваем GARCH(1,1) на a:**

**model = garch('GARCHLags',1,'ARCHLags',1);**

**[estM1,~,~] = estimate(model, a, 'Display','off'); % mean=0**

**% 3) Прогноз дисперсии одного шага вперёд:**

**HFor = forecast(estM1, 1, 'Y0', a); % HFor — это σ²\_{T+1}**

**% 4) Прогноз цены на следующий день:**

**price25\_hat = prices(end) + HFor;**

**fprintf('Прогноз дисперсии σ²\_{T+1} = %.6f\n', HFor);**

**fprintf('Прогноз цены y\_{T+1} = %.4f\n', price25\_hat);**

**Прогноз дисперсии σ²\_{T+1} = 0.098170**

**Прогноз цены y\_{T+1} = 45.0082**

**>> %% 1) Ваши цены по дням 1:24**

**prices = [45.27, 45.34, 45.51, 45.55, 45.17, 45.35, 45.43, 45.32, ...**

**44.72, 44.83, 44.78, 44.62, 44.98, 45.60, 46.01, 45.72, ...**

**45.84, 46.14, 46.16, 45.52, 45.05, 44.91, 45.17, 44.91]';**

**%% 2) Считаем приращения (mean=0 для GARCH)**

**a = diff(prices); % длина 23**

**%% 3) Задаём и оцениваем GARCH(1,1)**

**model = garch('GARCHLags',1,'ARCHLags',1);**

**[estM1,~,~] = estimate(model, a, 'Display','off');**

**%% 4) Прогноз дисперсии на 1 шаг вперёд**

**HFor = forecast(estM1, 1, 'Y0', a); % σ²\_{25}**

**%% 5) Прогноз цены на 25-й день**

**price24 = prices(end);**

**price25\_hat = price24 + HFor;**

**fprintf('Прогноз σ²\_{25} = %.6f\n', HFor);**

**fprintf('Прогноз цены y\_{25} = %.4f\n', price25\_hat);**

**%% 6) График исторических цен и одношагового прогноза**

**figure;**

**plot(1:24, prices, '-o','LineWidth',1.2); hold on;**

**scatter(25, price25\_hat, 80, 'r','filled');**

**xlabel('День торгов'); ylabel('Цена');**

**title('История цен и прогноз на 25-й день');**

**legend('Исторические цены','Прогноз y\_{25}','Location','Best');**

**grid on;**

**%% 7) График условной волатильности (σ\_t) по GARCH**

**% Получим все conditional\_volatility**

**[~,~,v] = infer(estM1, a);**

**sigma\_t = sqrt(v);**

**figure;**

**plot(2:24, sigma\_t, '-b','LineWidth',1.2); hold on;**

**plot(25, sqrt(HFor), 'rp','MarkerSize',12,'MarkerFaceColor','r');**

**xlabel('День торгов'); ylabel('σ\_t');**

**title('Условная волатильность GARCH(1,1)');**

**legend('σ\_t на обучении','Прогноз σ\_{25}','Location','Best');**

**grid on;**

**Прогноз σ²\_{25} = 0.098170**

**Прогноз цены y\_{25} = 45.0082**

**Python**

# %%

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import chi2

from statsmodels.graphics.tsaplots import plot\_pacf

from statsmodels.tsa.ar\_model import AutoReg

from statsmodels.tsa.stattools import adfuller

# %% [markdown]

# В-7.

#

#

# %%

crona = [45.27,45.34,45.51,45.55,45.17,45.35,45.43,45.32,44.72,44.83,44.78,44.62,

         44.98,45.60,46.01,45.72,45.84,46.14,46.16,45.52,45.05,44.91,45.17,44.91]

n= len(crona)

print(n)

# %%

def auto\_covar(crona, lag):

  mean = np.mean(crona)

  acov = [float(sum((crona - mean) \*\* 2) / len(crona))]

  for i in range(1, lag + 1):

    acov.append(float(sum((crona[i:] - mean) \* (crona[:-i] - mean)) / len(crona)))

  return acov

def auto\_corel(crona, lag):

  mean = np.mean(crona)

  acorr = [1]

  for i in range(1, lag + 1):

    acorr.append(float(sum((crona[i:] - mean) \* (crona[:-i] - mean)) / sum((crona - mean) \*\* 2)))

  return acorr

# %%

auto\_corel(crona, 10)

# %%

r = auto\_corel(crona, 10) # Q-статистика Льюнга-Бокса

r2 = [x \*\* 2 for x in r]

Q = 0

for k in range(1, 11):

  Q += n \* (n + 2) \* r2[k] / (n - k)

print(f'Q = {Q}')

k = 5

chi2\_critical = chi2.ppf(0.95, k)

if Q < chi2\_critical:

  print('H0 принимается')

else:

  print('H0 отвергается')

# %%

chi2\_critical

# %%

def Yule\_Walker(crona, p): #оценка Юла-Уокера

  r = auto\_corel(crona, p)[1:]

  R = np.ones((p, p))

  for i in range(p):

    for j in range(p):

      if i != j:

        R[i][j] = r[abs(i - j) - 1]

  a = np.linalg.solve(R, r)

  a0 = float(np.mean(crona) \* (1 - np.sum(a)))

  return np.array([a0, \*a])

print(Yule\_Walker(crona, 1))

# %%

ARp1 = Yule\_Walker(crona, 1)

pred\_ARp1 = crona[:1] + [ARp1[0] + ARp1[1] \* i for i in crona[:-1]]

# %% [markdown]

# Критические точки tкр распределения Дики-Фулера

# %%

y = crona[1:] # критерий Дарбина- Уотсона

X = np.column\_stack((np.ones(len(y)), crona[:-1]))

# (X^T X)^(-1)

XtX\_inv = np.linalg.inv(X.T @ X)

# Ищем c11 — элемент на позиции [1,1] (второй по счёту, т.к. первый — это смещение)

c11 = XtX\_inv[1, 1]

# Теперь можно вычислить стандартную ошибку s

n = len(y)

s = np.sqrt(c11 / (n - 1))

# Используем коэффициент при лаге (a1)

a1 = ARp1[1]

# ADF-статистика / критерий Дики-Фулера

gamma = (a1 - 1) / s

print("ADF-статистика γ =", gamma)

# %%

# Yule-Walker коэфы:

beta = Yule\_Walker(crona, 2)

# Формируем X и y для AR(2)

y = crona[2:]

x1 = crona[1:-1]

x2 = crona[0:-2]

X = np.column\_stack((np.ones(len(y)), x1, x2))

# Вычисляем (X^T X)^-1

XtX\_inv = np.linalg.inv(X.T @ X)

# Берём элемент c11 (для лаг-1, то есть второго столбца)

c11 = XtX\_inv[1, 1]

# Размер выборки

n = len(y)

# Стандартная ошибка s

s = np.sqrt(c11 / (n - 1))

# Берём a₁ из Yule-Walker

a1 = beta[1]

# Считаем ADF-статистику

gamma = (a1 - 1) / s

print("ADF-статистика γ =", gamma)

# %%

e2 = [(crona[i] - pred\_ARp1[i]) \*\* 2 for i in range(1, len(crona))] # критерий Акаике для p=1

AIC = 2 \* 1 / (len(crona) - 1) + np.log(sum(e2) \* 1 / (len(crona) - 1))

print(f'AIC = {AIC}')

# %%

ARp2 = Yule\_Walker(crona ,2)

pred\_ARp2 = crona[:2]+[ARp2[0] + ARp2[1]\*crona[i+1] + ARp2[2]\*crona[i] for i in range(len(crona[:-2]))]

e2 = [(crona[i] - pred\_ARp2[i]) \*\* 2 for i in range(2, len(crona))]

AIC = 2 \* 2 / (len(crona) - 2) + np.log(sum(e2) \* 2 / (len(crona) - 2))

print(f'AIC = {AIC}')

# %% [markdown]

# -6,15 < -5,43 => выбираем первую модель

# %%

# %%  -------- 1.3.1 – 1.3.3 : p = 1 … 5 ---------------------------------------

import numpy as np

from statsmodels.tsa.stattools import adfuller

def one\_step\_forecast(series, coeffs):

    """Одношаговый прогноз AR(p) с коэффициентами coeffs = (a0, φ1 … φp)."""

    p = len(coeffs) - 1

    y\_hat = list(series[:p])

    for t in range(p, len(series)):

        y\_hat.append(coeffs[0] + sum(coeffs[j] \* series[t-j] for j in range(1, p+1)))

    return np.array(y\_hat)

header = f'{"p":>2} | {"φ̂":18} | {"AIC":>7} | {"ADF":>7} | {"p-val":>6} | crit(5%)'

print(header)

print('-'\*len(header))

n = len(crona)

for p in range(1, 6):

    # --- 1.3.1  оценки Юла–Уокера -------------------------------------------

    coeffs = Yule\_Walker(crona, p)        # (a0, φ1 … φp)

    # --- 1.3.2  тест Дики–Фулера --------------------------------------------

    adf\_stat, p\_val, \_, \_, crit\_vals = adfuller(crona, maxlag=p, autolag=None)

    # --- 1.3.3  информационный критерий Акаике ------------------------------

    preds = one\_step\_forecast(crona, coeffs)

    sse   = np.sum((np.array(crona[p:]) - preds[p:])\*\*2)

    aic   = 2\*p/(n-p) + np.log(sse \* p/(n-p))

    print(f'{p:2d} | {np.round(coeffs[1:],3)}'.ljust(22),

          f'| {aic:7.3f} | {adf\_stat:7.3f} | {p\_val:6.3f} | {crit\_vals["5%"]:7.3f}')

# -----------------------------------------------------------------------------

# %%

# %% -------- AR(4) : оценка, подгонка, прогноз --------------------------------

import numpy as np

import matplotlib.pyplot as plt

p = 4

phi = Yule\_Walker(crona, p)            # (a0, φ1 … φ4)

print("\nОценённые коэффициенты AR(4):")

for i, c in enumerate(phi):

    print(f"{'a0' if i==0 else f'φ{i}'} = {c: .3f}")

# одномшаговый фит по всей выборке

y\_hat = one\_step\_forecast(crona, phi)

# прогноз на 25-й день (t = 24 — индекс последнего известного наблюдения 23)

y25 = phi[0] + sum(phi[j] \* crona[-j] for j in range(1, p+1))

print(f"\nПрогноз y₍₂₅₎ = {y25:.3f}")

# визуализация

plt.plot(range(1, len(crona)+1), crona, lw=2, label='факт')

plt.plot(range(1, len(crona)+1), y\_hat, '--', label='AR(4)')

plt.scatter(25, y25, color='red', zorder=5, label='прогноз 25-й')

plt.xlabel('день'); plt.ylabel('значение'); plt.legend(); plt.grid(True)

plt.show()

# %% [markdown]

# №2

# %%

print(f'Автокорреляция: {auto\_corel(crona,10)}')

print(f'Автоковариация: {auto\_covar(crona,10)}')

# %%

acf\_values = auto\_corel(crona, 10)

# Создаем фигуру

plt.figure(figsize=(12, 6))

# Рисуем столбцы автокорреляций (без use\_line\_collection)

markerline, stemlines, baseline = plt.stem(range(len(acf\_values)), acf\_values)

plt.setp(stemlines, 'linewidth', 2)  # Утолщаем линии

plt.setp(markerline, 'color', 'green')

# Подписи осей

plt.title('Коррелограмма (ACF)')

plt.xlabel('Лаг')

plt.ylabel('Значение автокорреляции')

plt.grid(True)

plt.show()

# %% [markdown]

# ARMA(p,1) невозможна, т.к. r1 >1/2

# %%

filtr\_X = [crona[i] - crona[i - 1] for i in range(2, len(crona))]

filtr\_X = filtr\_X - np.mean(filtr\_X)

filtr\_X

# %%

auto\_corel(filtr\_X,10)

# %%

teta1 = (-1 + np.sqrt(1-4 \* auto\_corel(filtr\_X, 10)[1] \*\* 2)) / (2 \* auto\_corel(filtr\_X, 10)[1])

teta2 =(-1 - np.sqrt(1-4 \* auto\_corel(filtr\_X, 10)[1] \*\* 2)) / (2 \* auto\_corel(filtr\_X, 10)[1])

print(f'teta1 = {np.round(teta1,3)}\nteta2 = {np.round(teta2,3)}')

# %%

auto\_covar(filtr\_X, 10)

# %%

param = [[auto\_covar(filtr\_X, 10)[0], auto\_covar(filtr\_X, 10)[1], auto\_covar(filtr\_X, 10)[2]],

 [auto\_covar(filtr\_X, 10)[1], auto\_covar(filtr\_X, 10)[0], auto\_covar(filtr\_X, 10)[1]],

 [auto\_covar(filtr\_X, 10)[2], auto\_covar(filtr\_X, 10)[1], auto\_covar(filtr\_X, 10)[0]]]

param

# %%

free\_members = [auto\_covar(filtr\_X, 10)[1], auto\_covar(filtr\_X, 10)[2], auto\_covar(filtr\_X, 10)[3]]

free\_members

# %%

ARIMA\_3\_1 = np.linalg.solve(param, free\_members)

ARIMA\_3\_1

# %%

error = [np.random.normal(0,1,1) for i in range(len(crona))] # Сгенерированные ошибки

error = list(np.concatenate(error))

print(f'Математическая модель: x(n) = {np.round(ARIMA\_3\_1[0],3)}\*\*x(n-1) {np.round(ARIMA\_3\_1[1],3)}\*x(n-2) {np.round(ARIMA\_3\_1[2],3)}\*x(n-3) + eps(n) {np.round(teta1,3)}\*eps')

# %%

predict\_ARIMA\_3\_1 = list(filtr\_X[:3]) + [ARIMA\_3\_1[0]\*filtr\_X[i-1] + ARIMA\_3\_1[1]\*filtr\_X[i-2] + ARIMA\_3\_1[2]\*filtr\_X[i-3] - teta1\*error[i-3] for i in range(3, len(filtr\_X)+1)]

# %%

plt.plot(range(1, len(filtr\_X)+1), filtr\_X, label="Сглаженные данные")

plt.plot(range(1, len( predict\_ARIMA\_3\_1[:-1])+1), predict\_ARIMA\_3\_1[:-1], label="Модель ARIMA(3,1)")

plt.legend()

# %%

auto\_corel(filtr\_X, 10)

# %%

from scipy import stats #Статистика Бокса-Пирса

m = 1

gamma = len(crona)\*auto\_corel(filtr\_X, 10)[1]

chi2 = stats.chi2.ppf(0.95, df=len(crona)-m-3)

print(f'm = {m} -> {gamma} <= {chi2}: {gamma <= chi2}')

while gamma <= chi2 and m <= len(crona):

  m += 1

  gamma = len(crona)\*np.sum((auto\_corel(filtr\_X, i)[i]) for i in range(1, m+1))

  chi2 = stats.chi2.ppf(0.95, df=len(crona)-m-3)

  print(f'm = {m} -> {gamma} <= {chi2}: {gamma <= chi2}')

# %%

ARIMA\_1\_1 = auto\_covar(filtr\_X, 10)[1] / auto\_covar(filtr\_X, 10)[0]

param = [[auto\_covar(filtr\_X, 10)[0],  auto\_covar(filtr\_X, 10)[1]], [ auto\_covar(filtr\_X, 10)[1],  auto\_covar(filtr\_X, 10)[0]]]

free\_members = [auto\_covar(filtr\_X, 10)[1],  auto\_covar(filtr\_X, 10)[2]]

ARIMA\_2\_1 = np.linalg.solve(param,free\_members)

predict\_ARIMA\_1\_1 = list(filtr\_X[:1]) + [ARIMA\_1\_1\*filtr\_X[i-1] - teta1\*error[i-1] for i in range(1, len(filtr\_X))]

predict\_ARIMA\_2\_1 = list(filtr\_X[:2]) + [ARIMA\_2\_1[0]\*filtr\_X[i-1] + ARIMA\_2\_1[1]\*filtr\_X[i-2] -teta1\*error[i-2] for i in range(2, len(filtr\_X))]

# %%

T = len(filtr\_X) # Критерий Акаике

# количество параметров

p1, q1 = 1, 1   # ARIMA(1,1)

p2, q2 = 2, 1   # ARIMA(2,1)

p3, q3 = 3, 1   # ARIMA(3,1)

sse1 = np.sum((np.array(filtr\_X) - np.array(predict\_ARIMA\_1\_1))\*\*2)

sse2 = np.sum((np.array(filtr\_X) - np.array(predict\_ARIMA\_2\_1))\*\*2)

sse3 = np.sum((np.array(filtr\_X) - np.array(predict\_ARIMA\_3\_1[:-1]))\*\*2)

AIC1 = 2\*(p1+q1)/T + np.log(sse1/T)

AIC2 = 2\*(p2+q2)/T + np.log(sse2/T)

AIC3 = 2\*(p3+q3)/T + np.log(sse3/T)

print(f'AIC1 = {np.round(AIC1,3)}\nAIC2 = {np.round(AIC2,3)}\nAIC3 = {np.round(AIC3,3)}')

# %% [markdown]

#  ARIMA(2,1) (-2.668) — лучшая модель среди трёх, так как у неё самый маленький AIC.

# %%

predict = predict\_ARIMA\_2\_1[-1] + crona[-1]

plt.plot(range(1,len(crona)+1), crona, label="Исходные данные")

plt.scatter(25, predict, color="green",label="Прогноз")

plt.legend()

print(f'Прогноз =  {predict}')

# %% [markdown]

# №3

# %%

prir = [(crona[i]-crona[i-1])/crona[i-1] for i in range(1, len(crona))]

prir

# %%

def auto\_covar\_k(k, crona):

  mean = np.sum(crona)/len(crona)

  return np.dot((crona[:len(crona)-k]-mean),(crona[k:]-mean))/len(crona)

# %%

# №2. Построение ARMA(p,1) для crona — исправленный вариант

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import chi2

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.stats.diagnostic import acorr\_ljungbox

# Данные

crona = np.array([

    45.27,45.34,45.51,45.55,45.17,45.35,45.43,45.32,

    44.72,44.83,44.78,44.62,44.98,45.60,46.01,45.72,

    45.84,46.14,46.16,45.52,45.05,44.91,45.17,44.91

])

n = len(crona)

# Вспомогательные функции

def auto\_covar(x, lag):

    μ = x.mean()

    cov0 = np.sum((x-μ)\*\*2)/n

    return [cov0] + [

        np.sum((x[k:]-μ)\*(x[:-k]-μ))/n

        for k in range(1, lag+1)

    ]

def auto\_corel(x, lag):

    μ = x.mean()

    den = np.sum((x-μ)\*\*2)

    return [1.0] + [

        np.sum((x[k:]-μ)\*(x[:-k]-μ)) / den

        for k in range(1, lag+1)

    ]

# 1. Автокорреляции до лага 10 + коррелограмма

cor = auto\_corel(crona, 10)

plt.figure(figsize=(8,4))

markerline, stemlines, baseline = plt.stem(range(len(cor)), cor)

plt.setp(stemlines, linewidth=1.5)

plt.setp(markerline, color='green')

plt.title("Коррелограмма (ACF) crona, лага 0–10")

plt.xlabel("Лаг")

plt.ylabel("ACF")

plt.grid(True)

plt.show()

p0 = 4

# 2. Центрируем данные и оцениваем ARMA(p0,1)

y\_c = crona - crona.mean()

model\_arma = ARIMA(y\_c, order=(p0, 0, 1)).fit()

print(model\_arma.summary())

# 3. Тест Бокса-Пирса по остаткам

resid = model\_arma.resid

lb = acorr\_ljungbox(resid, lags=10, boxpierce=True, return\_df=True)

print("\nBox–Pierce тест (χ² и p-value для каждого лага):")

print(lb[['bp\_stat','bp\_pvalue']])

# максимальное m с p-value > 0.05

m = (lb['bp\_pvalue'] > 0.05).idxmax() + 1

print(f"\nМаксимальное m с p-value>0.05: m = {m}")

# проверка стационарности по корням AR-полинома

roots = model\_arma.arroots

is\_stationary = np.all(np.abs(roots) > 1)

print("Стационарен ли процесс ARMA(p,1)?", is\_stationary)

# 4. Оптимальный p по AIC (перебор p=1…5)

aic\_list = []

for p\_test in range(1, 6):

    try:

        m\_tmp = ARIMA(y\_c, order=(p\_test, 0, 1)).fit()

        aic\_list.append((p\_test, m\_tmp.aic))

    except:

        pass

best\_p, best\_aic = min(aic\_list, key=lambda x: x[1])

print(f"\nОптимальный p по AIC = {best\_p}, AIC = {best\_aic:.3f}")

print("p из п.1 =", p0)

# 5. Прогноз на 25-й день методом ARMA(p0,1)

fc = model\_arma.get\_forecast(steps=1)

# predicted\_mean — numpy.ndarray, берем первый элемент

y25\_c = fc.predicted\_mean[0]

y25 = y25\_c + crona.mean()

print(f"\nПрогноз на 25-й день (ARMA({p0},1)): y₍₂₅₎ ≈ {y25:.3f}")

# 6. Если ARMA(p,1) не работает — строим ARIMA(best\_p,1,0)

try:

    model\_ari = ARIMA(crona, order=(best\_p, 1, 0)).fit()

    fc2 = model\_ari.get\_forecast(steps=1)

    y25\_ari = fc2.predicted\_mean[0]

    print(f"Прогноз на 25-й день (ARIMA({best\_p},1,0)): y₍₂₅₎ ≈ {y25\_ari:.3f}")

except Exception as e:

    print("ARIMA(p,1,0) не удалось оценить:", e)

# 8. Экспорт в CSV для проверки в Statistica

import pandas as pd

pd.DataFrame({'y': crona}).to\_csv('crona\_series.csv', index=False)

pd.DataFrame({'forecast\_25': [y25]}).to\_csv('crona\_forecast25.csv', index=False)

print("\nСохранены crona\_series.csv и crona\_forecast25.csv")

# %%

import numpy as np

from scipy.optimize import minimize

def estimate\_arima\_6\_1\_1(y):

    """

    Оценивает модель ARIMA(6,1,1) методом условных сумм квадратов без константы.

    Возвращает: массив AR-коэффициентов phi1..phi6, MA-коэффициент theta1 и вектор остатков.

    """

    # 1) Дифференцирование первого порядка

    dy = np.diff(y, n=1)

    dy = dy - np.mean(dy)

    T = len(dy)

    # 2) Функция CSS (Conditional Sum of Squares) без mu

    def css(params):

        phi = params[:6]    # phi1..phi6

        theta = params[6]   # theta1

        resid = np.zeros(T)

        for t in range(T):

            ar\_term = sum(phi[i] \* dy[t - i - 1]

                          for i in range(min(t, 6)))

            ma\_term = theta \* resid[t-1] if t > 0 else 0.0

            resid[t] = dy[t] - ar\_term - ma\_term

        return np.sum(resid\*\*2)

    # 3) Начальное приближение (phi=0..0, theta=0)

    init\_params = np.zeros(6 + 1)

    # 4) Оптимизация

    opt = minimize(css, init\_params, method='L-BFGS-B')

    phi\_hat = opt.x[:6]

    theta\_hat = opt.x[6]

    # 5) Вычисляем остатки по найденным phi и theta

    resid = np.zeros(T)

    for t in range(T):

        ar\_term = sum(phi\_hat[i] \* dy[t - i - 1]

                      for i in range(min(t, 6)))

        ma\_term = theta\_hat \* (resid[t-1] if t > 0 else 0.0)

        resid[t] = dy[t] - ar\_term - ma\_term

    return {

        'phi': phi\_hat,

        'theta': theta\_hat,

        'residuals': resid,

        'css': opt.fun

    }

# %%

y

# %%

np.diff(crona)

# %%

estimate\_arima\_6\_1\_1(crona)

# %%

resid = estimate\_arima\_6\_1\_1(y)

T = len(resid)

# Вычисляем ACF остатков и статистики Box–Pierce Q

lags = 10

acf = auto\_corel(resid['residuals'],10)

Q = [T \* sum(np.array(acf[:k])\*\*2) for k in range(1, lags+1)]

pvals = [chi2.sf(q, df=k) for k, q in enumerate(Q, start=1)]

# %%

pvals

# %%

import numpy as np

from scipy.optimize import minimize

import pandas as pd

def estimate\_css\_params(y, p):

    dy = np.diff(y, 1)

    dy = dy - np.mean(dy)

    T = len(dy)

    def css(params):

        phi = params[:p]

        theta = params[p]

        resid = np.zeros(T)

        for t in range(T):

            ar = sum(phi[i] \* dy[t - i - 1] for i in range(min(t, p)))

            ma = theta \* resid[t-1] if t > 0 else 0.0

            resid[t] = dy[t] - ar - ma

        return np.sum(resid\*\*2)

    init = np.zeros(p + 1)

    sol = minimize(css, init, method='L-BFGS-B')

    css\_val = sol.fun

    # Оценка AIC

    sigma2 = css\_val / T

    k = p + 1

    aic = T \* np.log(sigma2) + 2 \* k

    return css\_val, aic

def select\_p\_by\_aic(y, max\_p=10):

    results = []

    for p in range(max\_p + 1):

        css\_val, aic = estimate\_css\_params(y, p)

        results.append({'p': p, 'CSS': css\_val, 'AIC': aic})

    df = pd.DataFrame(results).set\_index('p')

    return df

# %%

select\_p\_by\_aic(crona, 7)

# %%

y = np.array([ 0.07,  0.17,  0.04, -0.38,  0.18,  0.08, -0.11, -0.6 ,  0.11,

              -0.05, -0.16,  0.36,  0.62,  0.41, -0.29,  0.12,  0.3 ,  0.02,

              -0.64, -0.47, -0.14,  0.26, -0.26])

phi = np.array([ 0.86970325, -0.40143956, -0.02909814,  0.46750285,

                -0.66175898, -0.07364059])

theta = -0.709529691506917

resid = np.array([ 0.08565217,  0.17193296,  0.05056559, -0.29985047,  0.28747433,

                   -0.04529167, -0.03858501, -0.26474524,  0.08244498, -0.26453032,

                   -0.1469147 ,  0.60461191,  0.22660197,  0.23638307, -0.15670525,

                    0.17876493,  0.16755782,  0.17580702,  0.0567955 , -0.06117996,

                   -0.10021193,  0.31434167,  0.01453486])

# Вычисление прогноза

dy = y

p = len(phi)

ar\_term = sum(phi[i] \* dy[-i-1] for i in range(p))

ma\_term = theta \* resid[-1]

delta\_forecast = ar\_term + ma\_term

y\_forecast = y[-1] + delta\_forecast

print("Прогноз Δy:", delta\_forecast)

print("Прогноз y:", y\_forecast)

# Построение графика

plt.figure()

plt.plot(np.arange(len(y)), y, label='Наблюдаемые значения')

plt.scatter([len(y)], [y\_forecast], label='Прогноз', zorder=5)

plt.legend()

plt.title('Одношаговый прогноз ARIMA(6,1,1)')

plt.xlabel('Индекс времени')

plt.ylabel('Значение')

plt.show()

# %%

crona

# %%

diffs =[]

for i in range(len(crona)-1):

    diffs.append((crona[i+1]- crona[i])/crona[i])

# %%

diffs = np.array(diffs)

# %%

diffs

# %%

results = []

for k in [5, 10, 15]:

    segment = diffs[-k:]

    mean\_k = segment.mean()

    var\_k = segment.var(ddof=0)  # смещённая оценка

    results.append({'k': k, 'mean': mean\_k, 'variance': var\_k})

# %%

results

# %%

# Параметры GARCH(1,1)

omega = 4.8e-05

alpha = 0.04

beta  = 0.89

T = len(diffs)

# Инициализация sigma^2

sigma2 = np.empty(T)

# Начальное значение взято условной несмещённой дисперсии ω/(1-α-β)

sigma2[0] = omega / (1 - alpha - beta)

# Рекуррентное вычисление σ²\_t

for t in range(1, T):

    sigma2[t] = omega + alpha \* diffs[t-1]\*\*2 + beta \* sigma2[t-1]

# Серии для анализа

u2      = diffs\*\*2

std\_u2  = u2 / sigma2

# Функция расчёта автокорреляции

def acf(x, lag):

    return np.corrcoef(x[:-lag], x[lag:])[0,1]

lags = list(range(1,6))

acf\_u2     = [acf(u2,      k) for k in lags]

acf\_std\_u2 = [acf(std\_u2,  k) for k in lags]

# Собираем в DataFrame

df = pd.DataFrame({

    'lag': lags,

    'ACF(u^2)': acf\_u2,

    'ACF(u^2/σ^2)': acf\_std\_u2

})

# %%

sigma2

# %%

df.values

# %%

# %%

import numpy as np

import matplotlib.pyplot as plt

from arch import arch\_model

# 1. Задаём модель GARCH(1,1) с нормальным распределением ошибок

am = arch\_model(diffs, vol='Garch', p=1, q=1, dist='normal')

# 2. Оцениваем модель

res = am.fit(disp='off')

# 3. Смотрим результаты

print(res.summary())

# 4. Строим условную волатильность

plt.figure()

plt.plot(res.conditional\_volatility)

plt.title('Условная волатильность GARCH(1,1)')

plt.xlabel('Индекс времени')

plt.ylabel('σₜ')

plt.show()

# %%

df

# %%

lags = df.values[:,0].astype(int)

acf\_u2 = df.values[:,1]

acf\_std = df.values[:,2]

T = 23  # длина рядов

# Функция для вычисления статистики Ljung-Box

def ljung\_box(acf, T, m):

    # Q = T\*(T+2) \* sum\_{k=1}^m acf\_k^2/(T-k)

    return T\*(T+2) \* np.sum(acf[:m]\*\*2 / (T - lags[:m]))

# Вычисляем Q и p-values для m = 5

m = 5

Q\_u2 = ljung\_box(acf\_u2, T, m)

Q\_std = ljung\_box(acf\_std, T, m)

pval\_u2 = 1 - chi2.cdf(Q\_u2, df=m)

pval\_std = 1 - chi2.cdf(Q\_std, df=m)

df = pd.DataFrame({

    'Series': ['u^2', 'u^2/σ^2'],

    'Q(m=5)': [Q\_u2, Q\_std],

    'p-value': [pval\_u2, pval\_std]

})

# %%

df

# %%

def auto\_covar\_k(k, crona):

  mean = np.sum(crona)/len(crona)

  return np.dot((crona[:len(crona)-k]-mean),(crona[k:]-mean))/len(crona)

# %%

corr1 = [] # Последовательность автокорреляций до порядка 5 включительно для ряда u^2(n)

for i in range(1,6):

  corr1.append(auto\_covar\_k(i, np.array(diffs)\*\*2)/auto\_covar\_k(0, np.array(diffs)\*\*2))

  print(f"r{i} = {corr1[i-1]}")

np.array(corr1)

# %%

prir2 = [diffs[i]\*\*2/sigma2[i] for i in range(1,len(prir))] # Последовательность u^2(n)/sigma^2(n)

corr2 = []

for i in range(1,6):

  corr2.append(auto\_covar\_k(i, prir2)/auto\_covar\_k(0, prir2))

  print(f"r{i} = {corr2[i-1]}")

np.array(corr2)

# %%

T = 24  # длина рядов

# Функция для вычисления статистики Ljung-Box

def ljung\_box(acf, T, m):

    lags = np.arange(m)

    return T\*(T+2) \* np.sum(np.power(acf[:m],2)/ (T - lags[:m]))

# Вычисляем Q и p-values для m = 5

m = 5

Q\_u2 = ljung\_box(corr1, T, m)

Q\_std = ljung\_box(corr2, T, m)

pval\_u2 = 1 - chi2.cdf(Q\_u2, df=m)

pval\_std = 1 - chi2.cdf(Q\_std, df=m)

df = pd.DataFrame({

    'Series': ['u^2', 'u^2/σ^2'],

    'Q(m=5)': [Q\_u2, Q\_std],

    'p-value': [pval\_u2, pval\_std]})

# %%

df

# %%

# Параметры вашей GARCH(1,1)

omega = 4.8e-06

alpha = 0.01

beta  = 0.89

T = len(diffs)

# Инициализируем массив условной дисперсии

sigma2 = np.empty(T)

# Берём стационарное начальное значение

sigma2[0] = omega / (1 - alpha - beta)

# Рассчитываем σ²\_t по рекурсии

for t in range(1, T):

    sigma2[t] = omega + alpha \* diffs[t-1]\*\*2 + beta \* sigma2[t-1]

# Одноступенчатый прогноз условной дисперсии σ²\_{T+1}

sigma2\_next = omega + alpha \* diffs[-1]\*\*2 + beta \* sigma2[-1]

print(f"Последняя оценка σ²\_T = {sigma2[-1]:.8f}")

print(f"Прогноз σ²\_{{T+1}} = {sigma2\_next:.8f}")

# %%

omega, alpha, beta = 4.8e-06, 0.01, 0.89

# рассчёт прогнозной дисперсии σ²\_{T+1}

pred\_sigma2 = omega + alpha \* corr1[-1] + beta \* prir2[-1]

# прогноз цены как y\_T плюс прогноз дисперсии

predict = crona[-1] + pred\_sigma2

print(f'Прогноз цены y\_(T+1) = {predict}')

# %%

plt.plot(crona)

plt.scatter(25, predict)

plt.title("Предсказание модели GARCH(1,1)")

# %%

crona