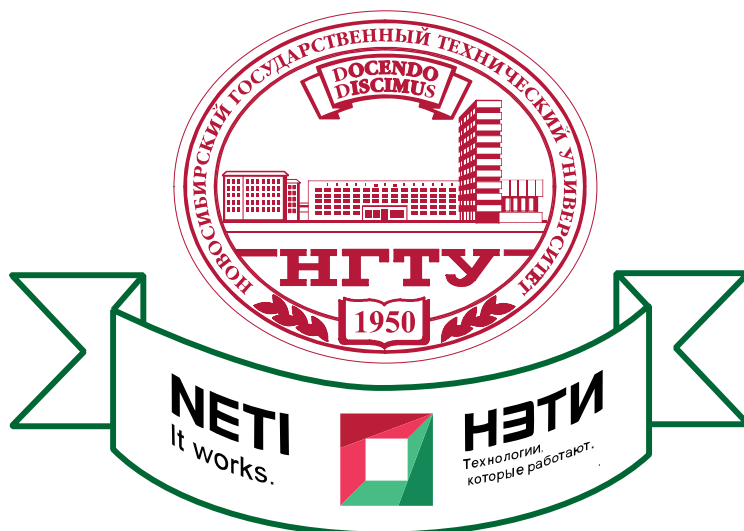


Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

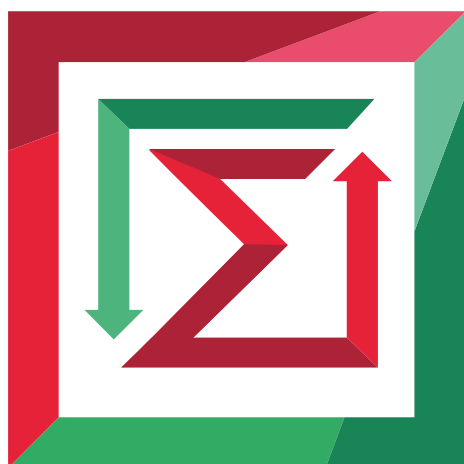
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Теоретической и прикладной информатики

Расчетно-графическая задача
по дисциплине «Компьютерное моделирование»

ПРОГНОЗ ВРЕМЕННОГО РЯДА МЕТОДОМ SSA



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

Новосибирск

2026

1. Формулировка задания

Выполнить прогноз временного ряда рекуррентным методом SSA (Singular Spectrum Analysis, "Гусеница") на M значений вперед. В качестве обучающей выборки использовать N значений временного ряда.

Для реализации метода SSA необходимо задать 2 параметра: L - ширина окна для построения траекторного пространства ряда, r - количество компонент, используемых для прогноза ($r < L < N$).

2. Описание алгоритма

SSA (Singular Spectrum Analysis) — метод анализа временных рядов, основанный на использовании линейных комбинаций собственных векторов матрицы корреляции. Алгоритм SSA можно разделить на следующие этапы:

1. Формирование траекторной матрицы (Trajectory Matrix, X). Входной временной ряд разбивается на окна фиксированной длины, и на каждом окне формируется вектор-траектория длины L (где L - длина окна). Все эти траектории объединяются в матрицу размера $L \times K$, где K - количество окон.
2. Разложение траекторной матрицы на сингулярные значения (Singular Value Decomposition, SVD). Применяется SVD, позволяющее представить матрицу в виде произведения трёх матриц: $X = U * S * V^T$, где U и V – ортогональные матрицы, а S – диагональная матрица с сингулярными значениями
3. Выбор главных компонент (Principal Components). Выбираются главные компоненты путем суммирования первых r сингулярных значений. Для этого строится матрица C размера $L \times K$, которая получается путем умножения первых r строк матрицы U на первые r столбцов матрицы V^T .
4. Формирование вложенных траекторий (Embedded Trajectories) из главных компонент. Каждый столбец матрицы C превращается в вектор-траекторию, и эти векторы объединяются в матрицу размера $(L - r + 1) \times r$, где r – количество выбранных главных компонент.
5. Расчет автокорреляционной функции (Autocorrelation Function, ACF) для каждой вложенной траектории. Это позволяет определить длину периода (периодичность) ряда и другие характеристики.

6. Выделение циклических компонент (Cyclic Components). Определяются вложенные траектории, чья длина периода соответствует периоду исходного временного ряда. Эти вложенные траектории объединяются в матрицу размера $L \times P$, где P – количество циклических компонент.
7. Выделение тренда и шума. Остальные вложенные траектории считаются шумом, а тренд определяется как среднее значение всех траекторий, не являющихся шумом.
8. Восстановление временного ряда. Итоговый временной ряд получается путем суммирования циклических компонент и тренда. Для прогнозирования значений временного ряда на будущих временных интервалах к тренду добавляются прогнозы циклических компонент на эти интервалы.

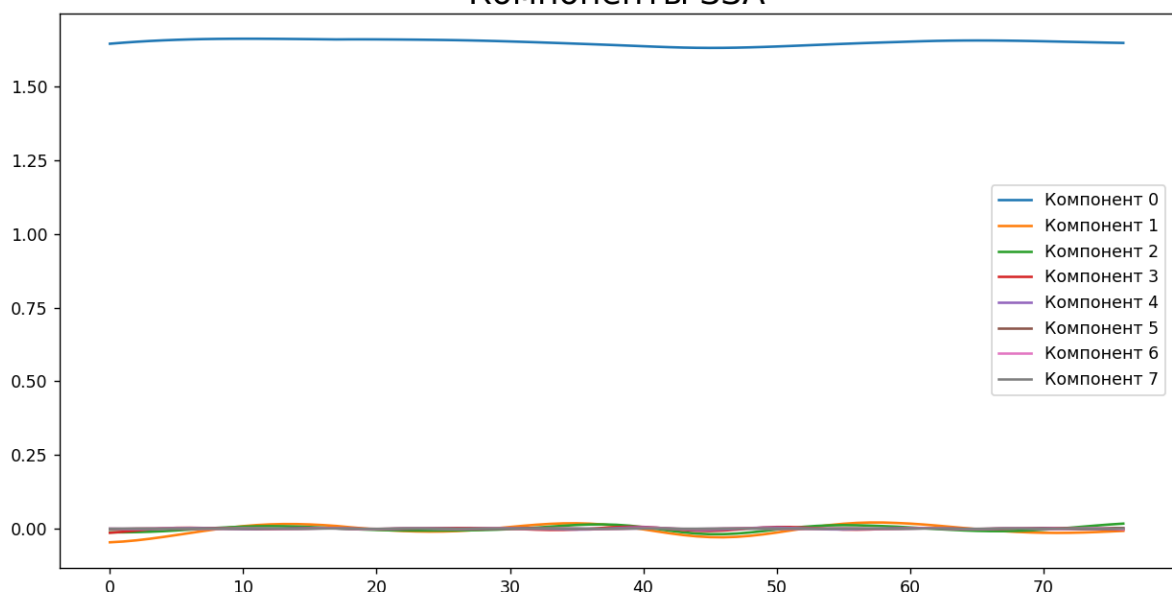
3. Ход работы

Временной ряд №1

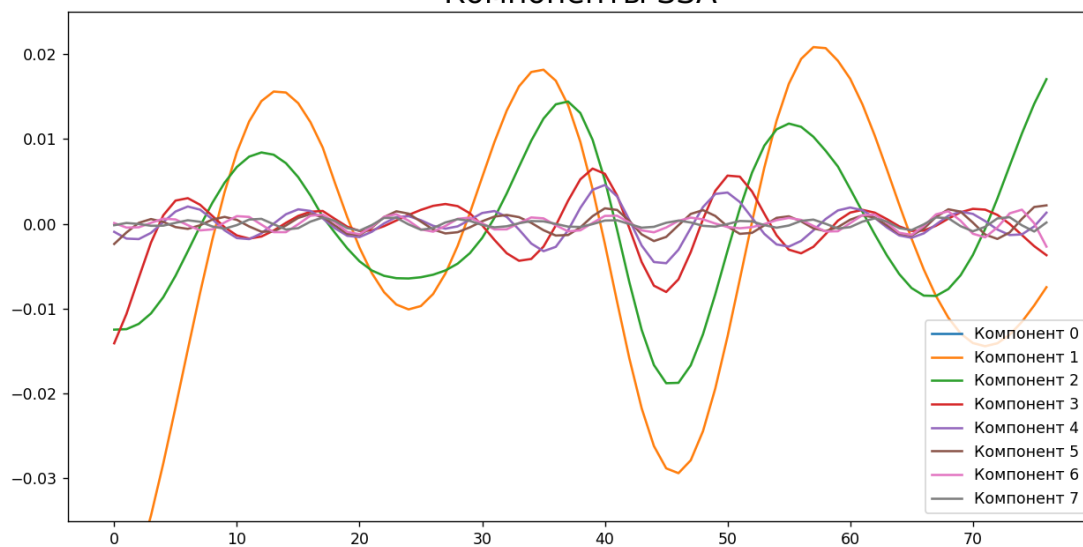
Протестируем прогнозирование при различных входных данных:

N	L	r	M
87	60	8	10

Компоненты SSA



Компоненты SSA



SSA прогноз

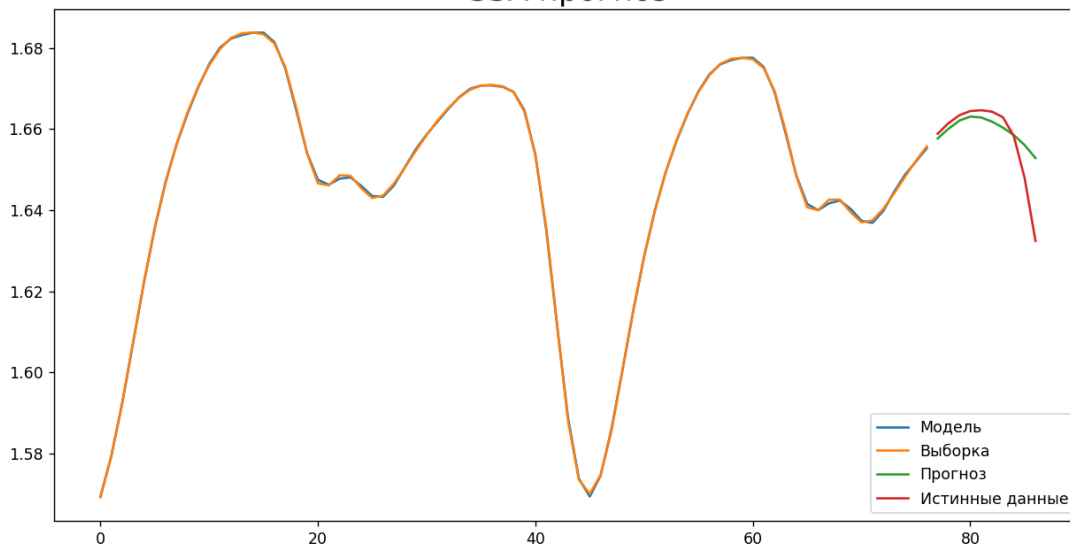
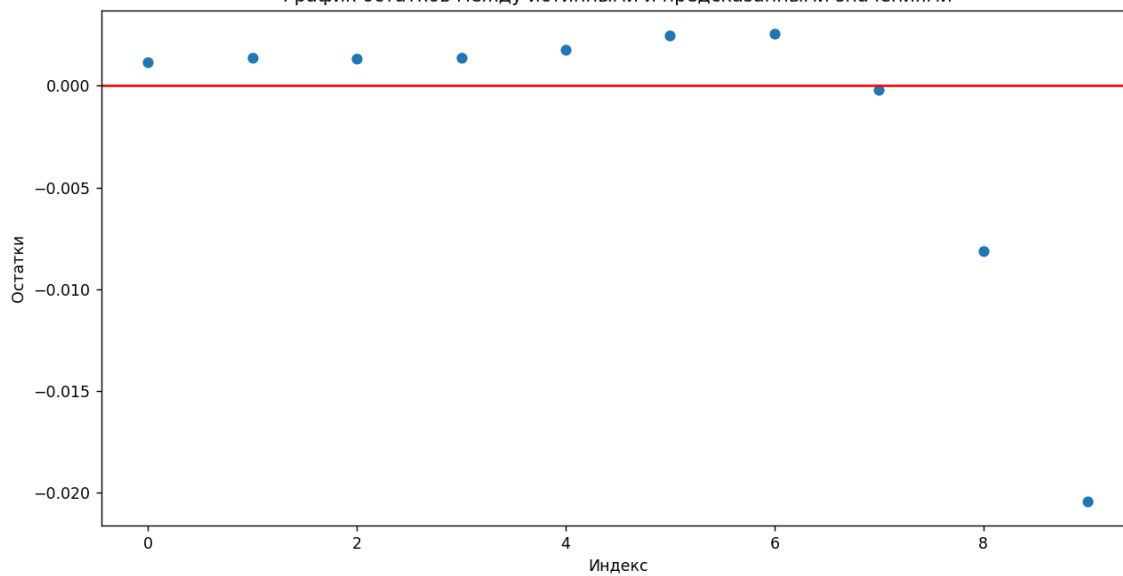
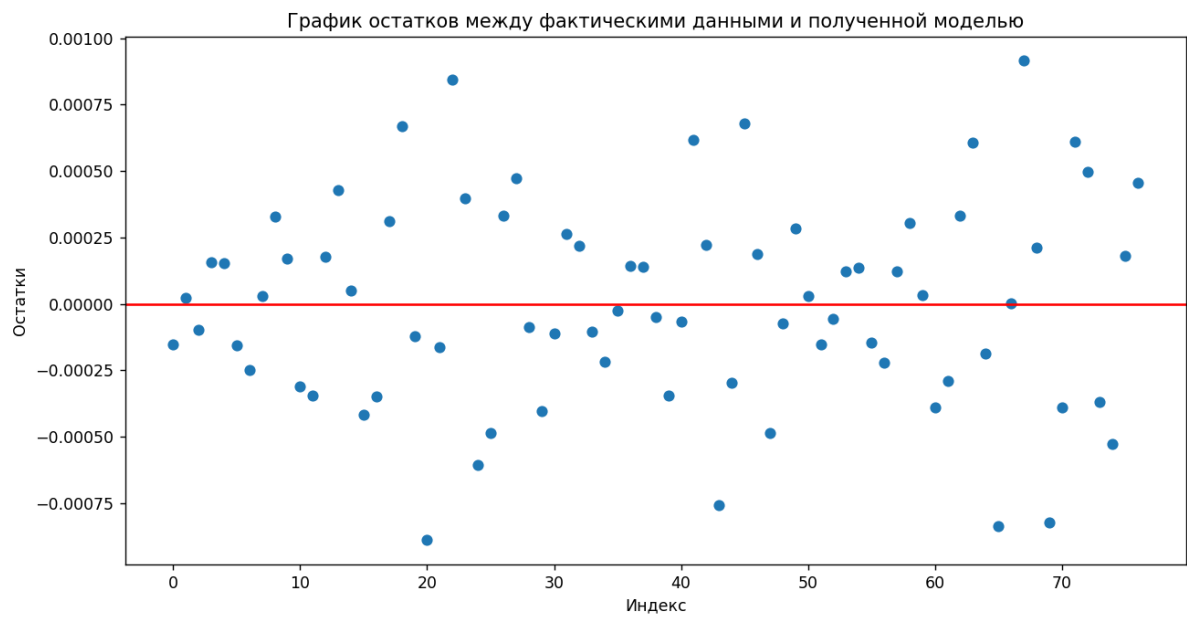


График остатков между истинными и предсказанными значениями

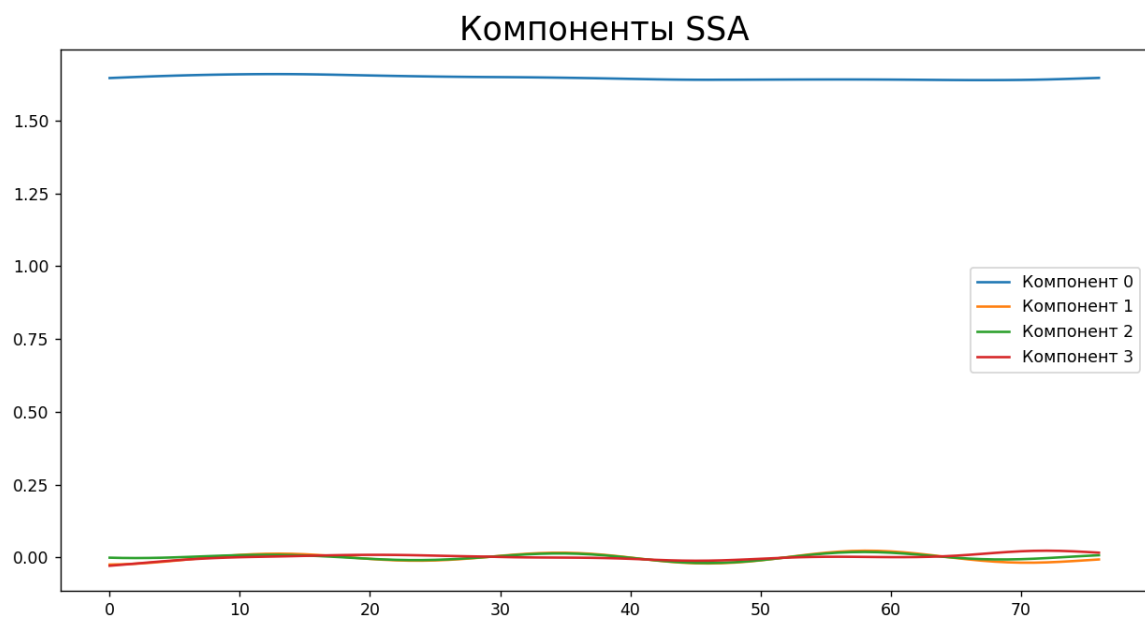


RMSE: 0.007115349606647277

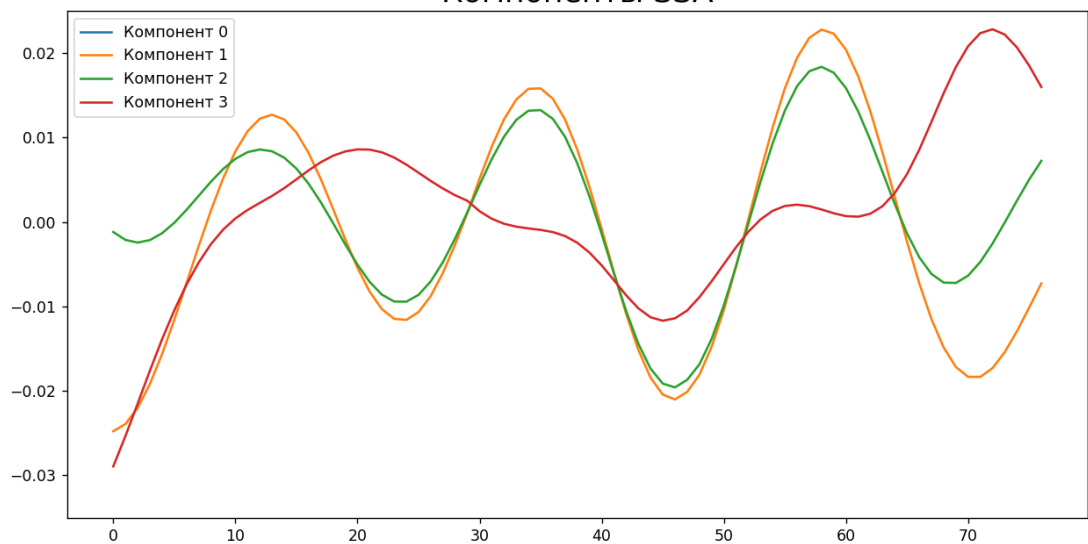


RMSE: 0.0003835699414160156

N	L	r	M
87	30	4	10



Компоненты SSA



SSA прогноз

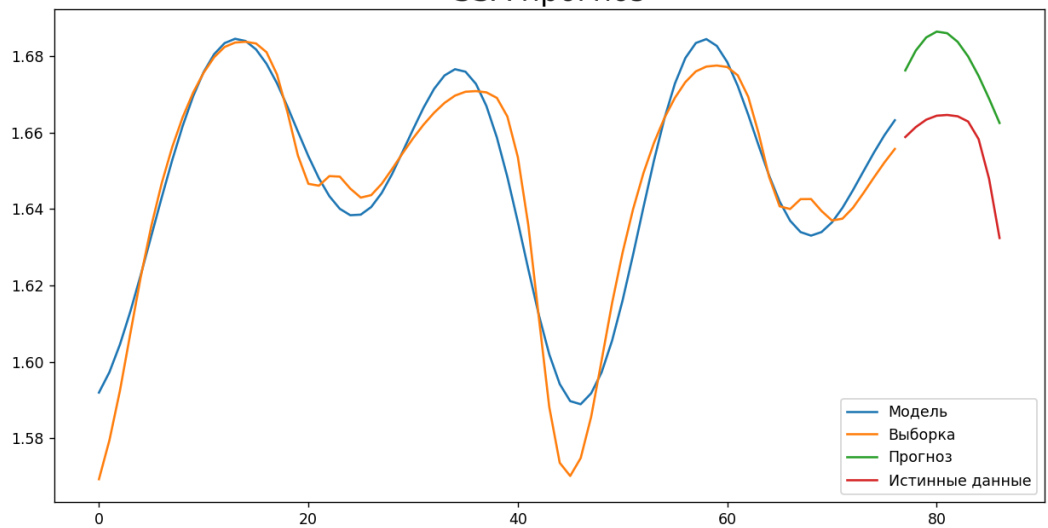
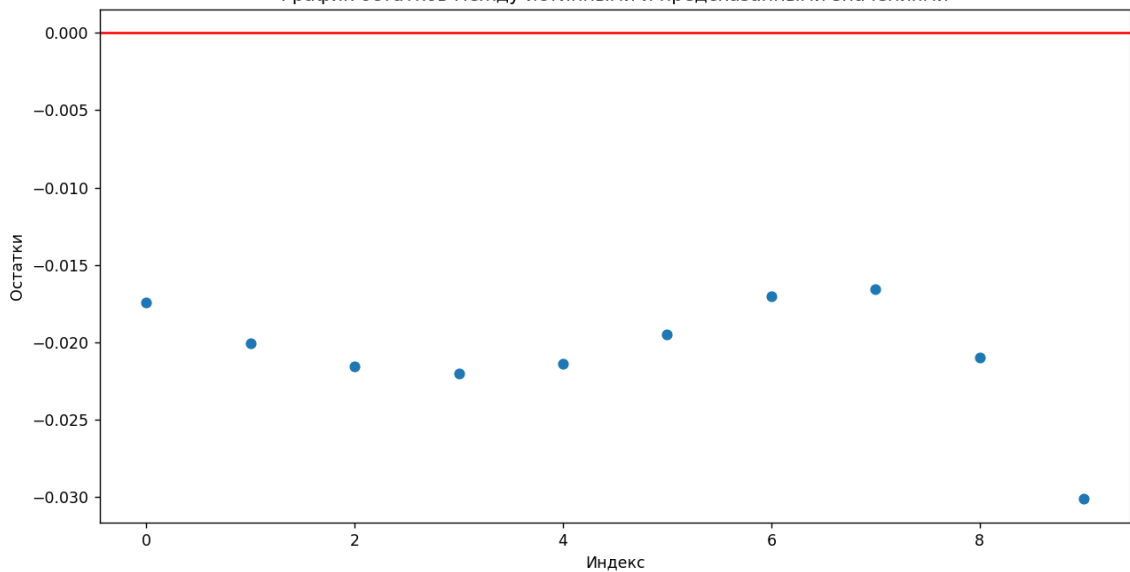
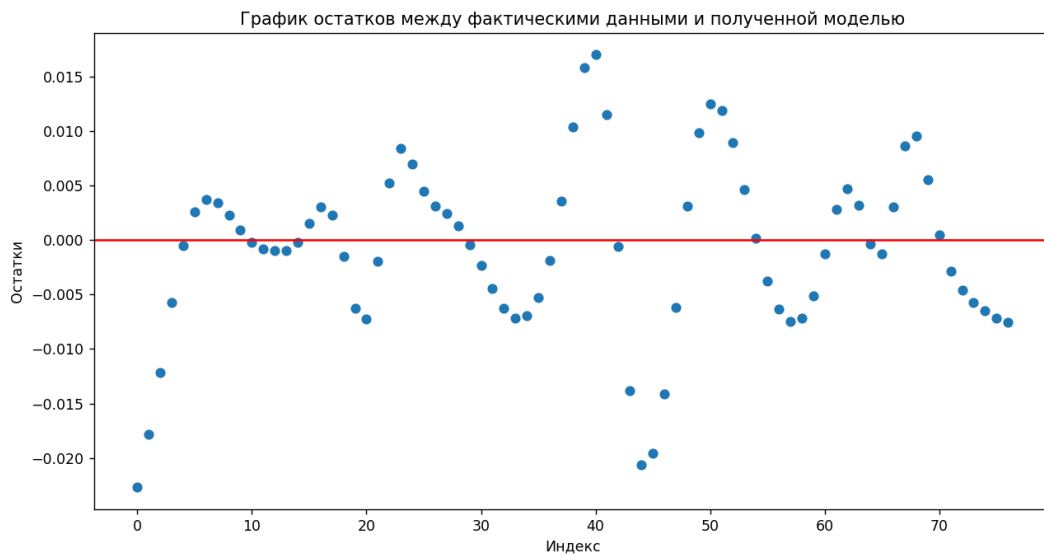


График остатков между истинными и предсказанными значениями



RMSE: 0.02099465106626816

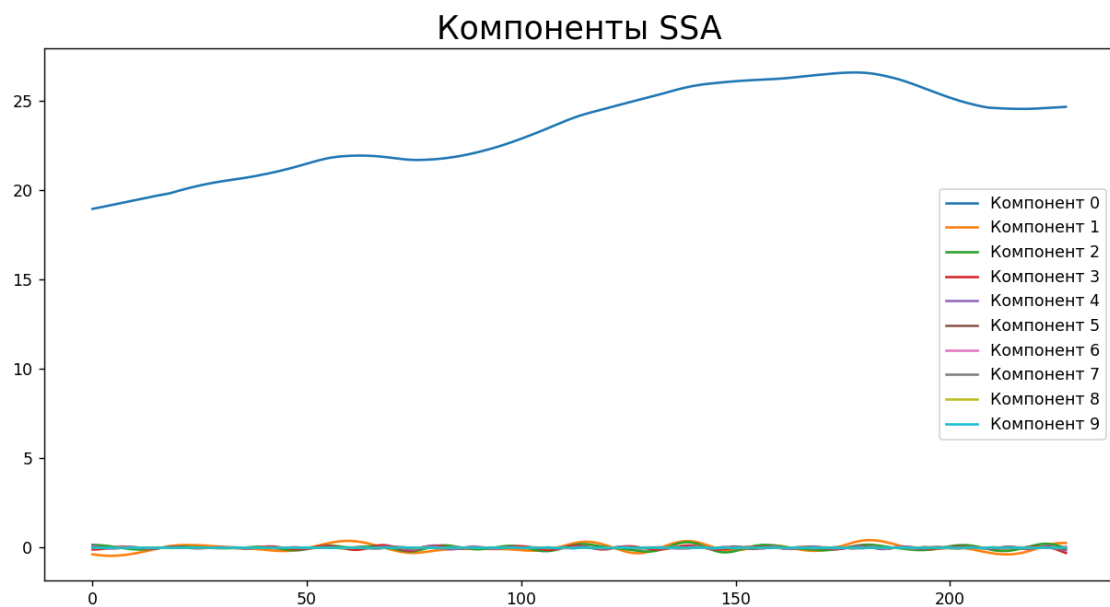


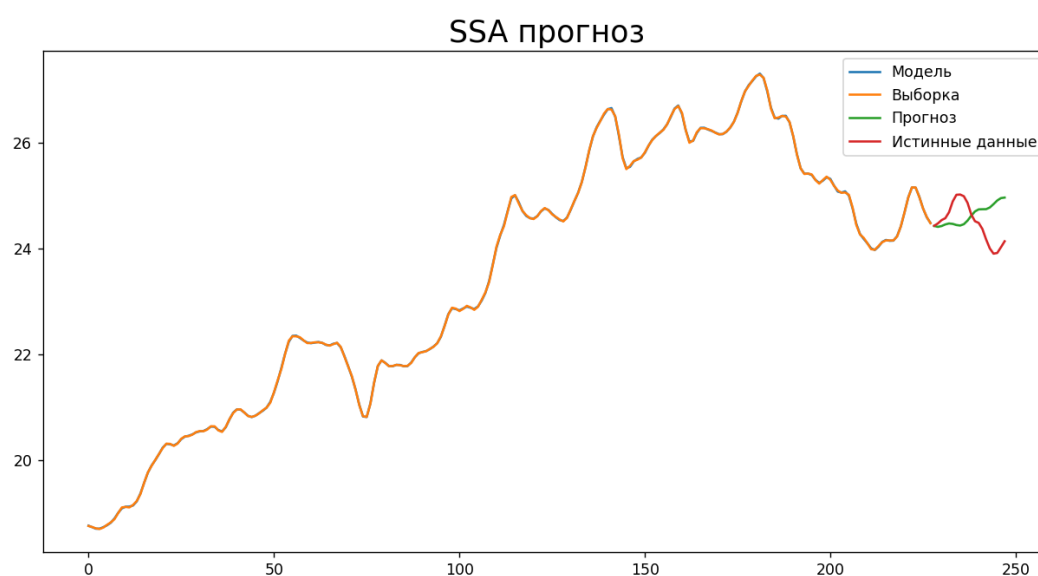
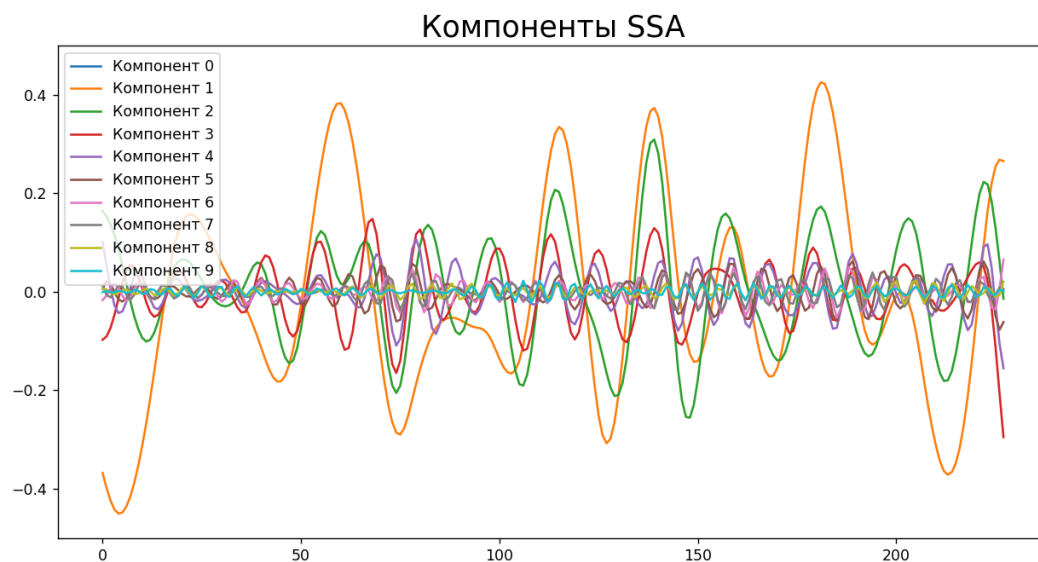
RMSE: 0.00784235181812238

Видим, что, уменьшив ширину окна и количество компонент, прогноз стал хуже. Остатки между моделью и фактическими данными имеют неслучайный характер. Сделаем вывод, что можно и необходимо выделить больше компонент, чтобы остатки имели нормальное распределение.

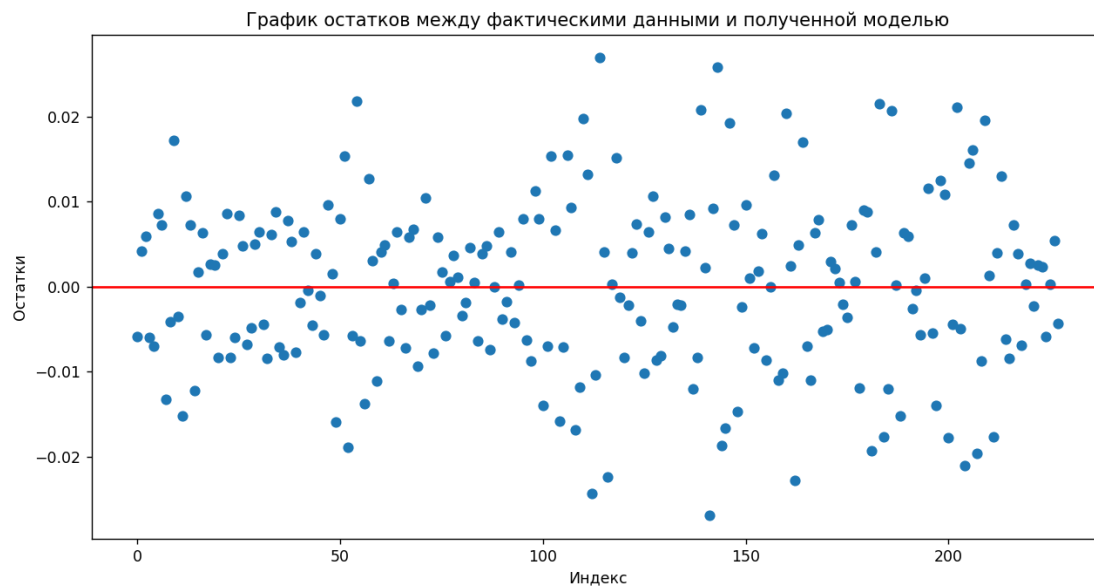
Временной ряд №2 (Курс валют)

N	L	r	M
248	210	10	20



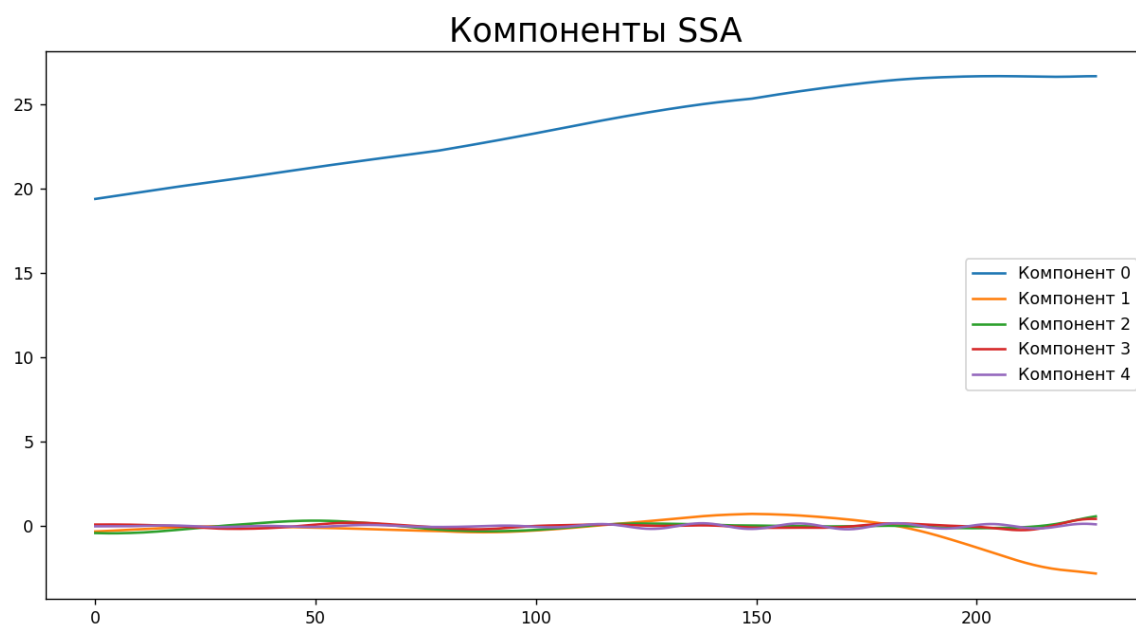


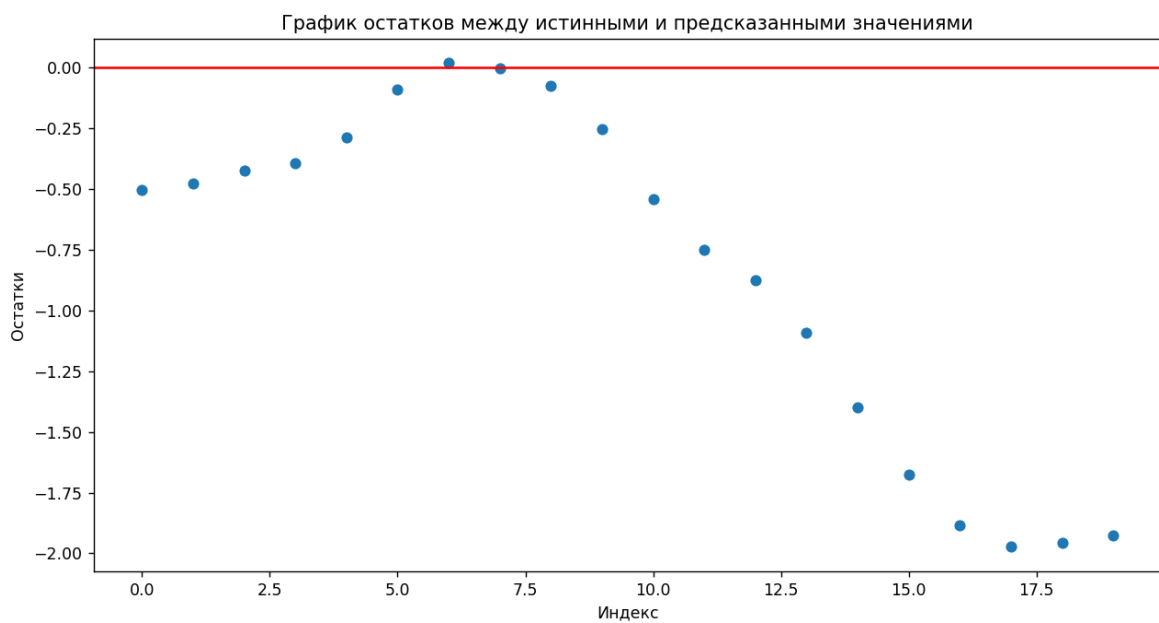
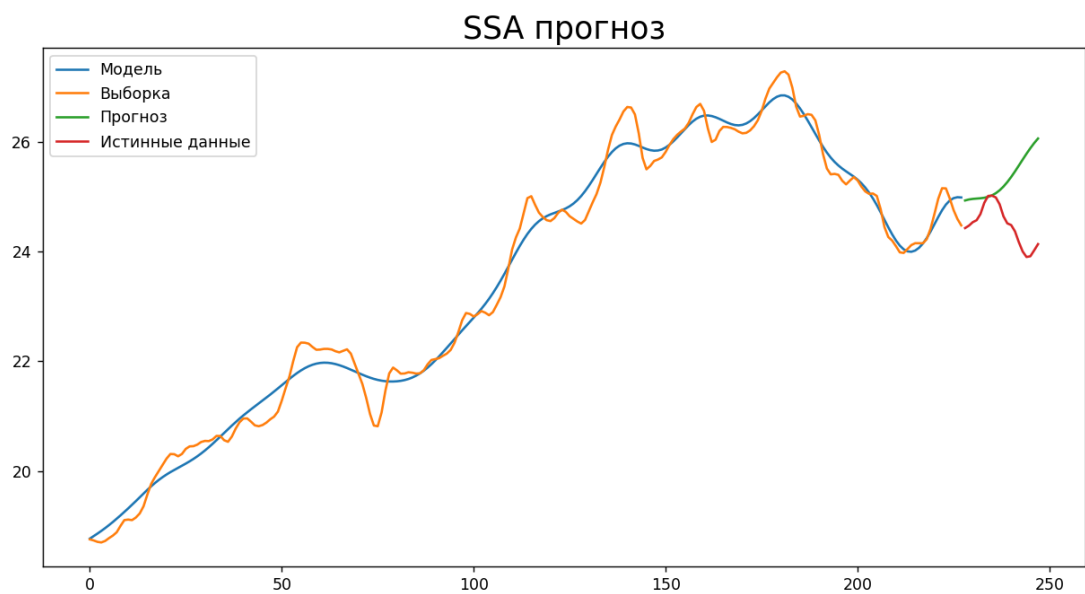
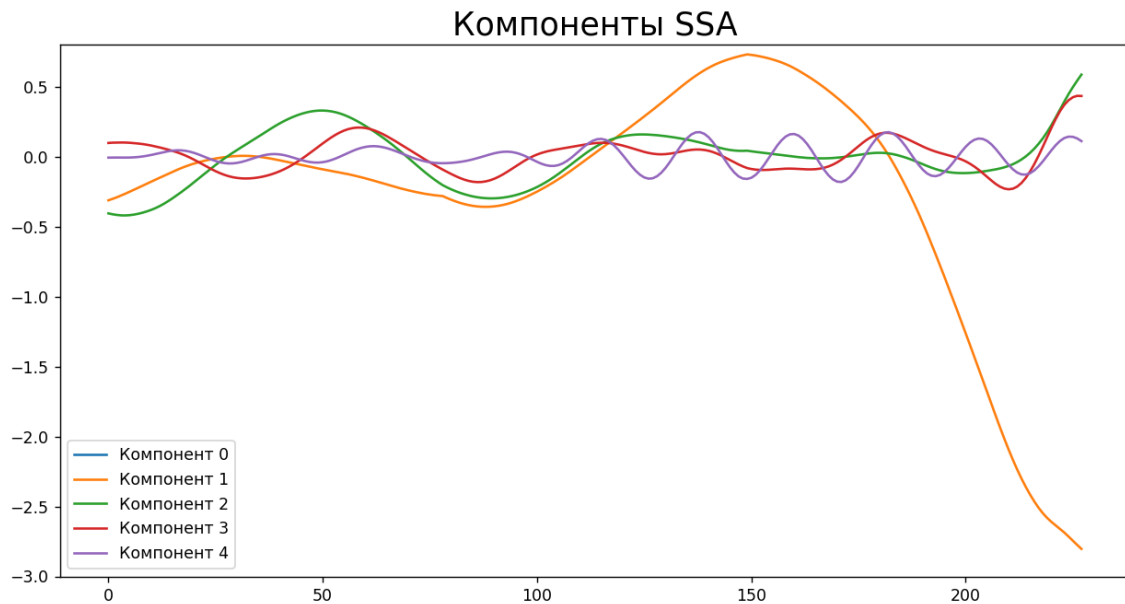
RMSE: 0.5431774805350388



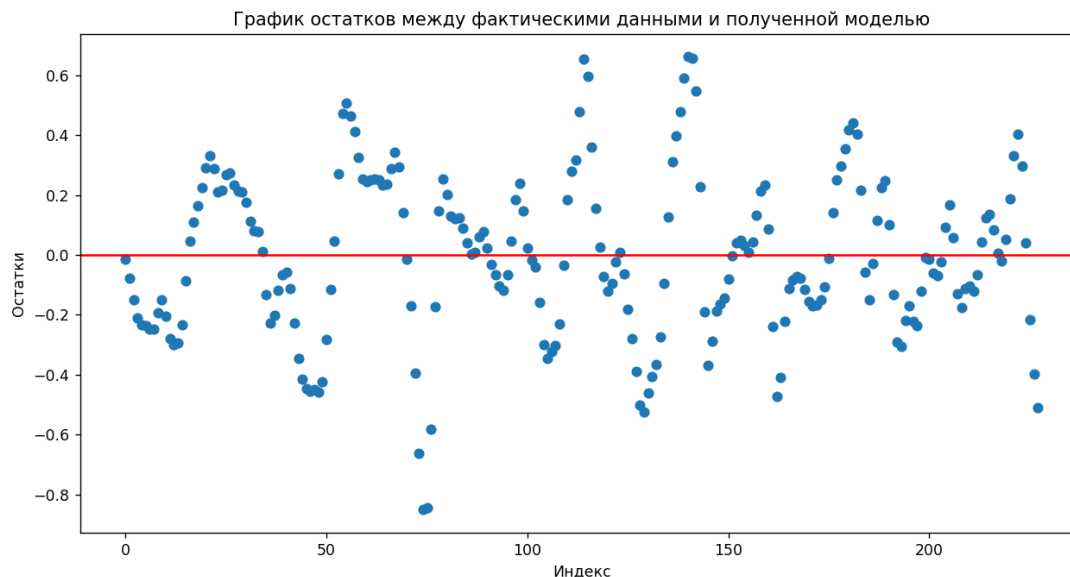
RMSE: 0.01004325982630331

N	L	r	M
248	150	5	20





RMSE: 1.0855858990637848



RMSE: 0.2701006660127868

Снова замечаем, что, уменьшив ширину окна и количество компонент, прогноз стал хуже. Остатки между моделью и фактическими данными имеют неслучайный характер. Сделаем аналогичный вывод о том, что можно и необходимо выделить больше компонент, чтобы остатки имели нормальное распределение.

Заполним таблицу с разными параметрами моделей для двух выборок:

Данные №1:

N	L	r	RMSE
20	5	3	0.02408950421923902
	5	6	0.02408950421923902
	10	3	0.02224728111070782
40	15	4	0.008424453595393917
	15	6	0.01526229891804096
	20	8	0.007651540396588195
60	20	4	0.01911643114246058
	30	6	0.0219956248025709

	40	8	0.012784058710896697
67	30	4	0.02099465106626816
	35	6	0.0060082077871025106
	40	8	0.07666973627163057
	60	8	0.007115349606647277

Курс валют:

N	L	r	RMSE
60	20	10	1.3039661140373109
	30	5	1.246245308030433
	30	10	1.1596600035103535
120	50	5	1.3975160947221588
	70	10	0.5563222275442301
	85	15	0.47103180826520036
180	90	5	0.9380594757293415
	120	10	1.445433471926849
	145	15	0.6199897882808622
228	120	5	2.84498519497443
	150	10	0.891528988823395
	180	15	0.6858381259701175
	210	10	0.5431774805350388
	210	15	0.52974139012194

Наблюдаем, что с ростом параметров качество прогноза улучшается. Есть случаи, когда при низких параметрах делается вполне верное предсказание. Если сравнивать результаты этой модели и моделей из лабораторной работы №2, то по RMSE она не всегда лучше других, но если смотреть на график предсказанных значений, то, в отличие от других моделей, SSA строит более реалистичное и возможное предсказание.

4. Вывод

Сравнивая SSA с моделями, использованными в лабораторной работе №2, можно сделать вывод, что первый предсказывает заметно лучше наши временные ряды. Вероятно, это связано с тем, что SSA эффективен для анализа рядов с ярко выраженной сезонностью или изменяющимися трендами.

Количество компонент и ширина окна напрямую влияют на качество предсказания и построения модели. Остатки между моделью и фактическими данными имеют неслучайный характер, если выделены не все возможные компоненты.

5. Листинг программы

SSA.py:

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

N = 120 # Величина выборки
M = 20 # M значений вперед
r = 15 # r - количество компонент
L = 180 # L - ширина окна для построения траекторного пространства ряда

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

data = pd.read_excel(DATASET)
time_series = data["smoothed_data"]#.tail(N)

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()

```

core.py:

```

import pandas as pd
import numpy as np
from scipy import linalg
from scipy.stats import norm

class MSSA(object):
    """
    Multi-channel Singular Spectrum Analysis object
    SSA class take one positional argument - timeseries.
    :param timeseries: type can be pandas.DataFrame, pandas.Series, numpy.array,
    numpy.matrix, list
    """
    def __init__(self, time_series):
        self.ts_df = pd.DataFrame(time_series).reset_index(drop=True)
        self.ts = np.matrix(self.ts_df)
        print(self.ts)
        self.ts_N = self.ts.shape[0]
        print(self.ts_N)
        self.ts_s = self.ts.shape[1]
        print(self.ts_s)
        self.ts_name = self.ts_df.columns.tolist()

    def _hankelSeries(self, series):
        """
        Perform hankelization procedure for given series
        """
        return linalg.hankel(series, np.zeros(self.embedding_dimension)).T[:,
:self.K]

    def embed(self, embedding_dimension=None):
        """
        Compute the trajectory matrix of given time series.

```

:param embedding_dimension: How many components to compute from the original series.
 Default is $N//2$, where N stands for length of series.

```

"""
    if not embedding_dimension:
        self.embedding_dimension = self.ts_N // 2
    else:
        self.embedding_dimension = embedding_dimension
    self.K = self.ts_N - self.embedding_dimension + 1
    series = np.hsplit(self.ts, self.ts_s)
    X = np.hstack(list(map(self._hankelSeries, series)))
    self.X = np.matrix(X)
    self.trajectory_dimensions = X.shape

def decompose(self):
    """
    Perform the Singular Value Decomposition and identify the rank of the
    embedding subspace.
    """
    X = self.X
    self.S = X * X.T
    self.U, self.s, self.V = linalg.svd(self.S)
    self.U, self.s, self.V = np.matrix(self.U), np.sqrt(self.s),
np.matrix(self.V)
    self.d = np.linalg.matrix_rank(X)
    self.Vs = (X.T * self.U) / self.s
    U_s = np.matrix(np.array(self.U) * self.s)
    Xs = np.empty((self.embedding_dimension, 0))
    for i in range(self.d):
        Xs = np.hstack((Xs, U_s[:, i] * self.Vs[:, i].T))
    self.Xs = Xs

@staticmethod
def _diagonal_averaging(hankel_matrix):
    """
    Performs anti-diagonal averaging from given hankel matrix
    :param embedding_dimension: Trajectory matrix of one-dimensional time
series.
    :return: pandas.DataFrame with decomposed series.
    """
    mat = np.matrix(hankel_matrix)
    L, K = mat.shape
    L_star, K_star = min(L, K), max(L, K)
    if L > K:
        mat = mat.T
    ret = []
    # Diagonal Averaging
    for k in range(1 - K_star, L_star):
        mask = np.eye(K_star, k=k, dtype='bool')[::-1][:L_star, :]
        mask_n = sum(sum(mask))
        ma = np.ma.masked_array(mat.A, mask=1 - mask)
        ret += [ma.sum() / mask_n]
    return ret

def _d_series_diag(self, Xd):
    '''Diagonal averaging for d series matrix'''
    sseries = list(map(self._diagonal_averaging, np.hsplit(Xd, self.ts_s)))
    return np.hstack(sseries)

```

```

def diag_procedure(self):
    '''Performs anti-diagonal averaging for multidimensional time series.'''
    _hankel_list = np.hsplit(self.Xs, self.d)
    _big_vector = np.vstack(list(map(self._d_series_diag, _hankel_list))).T
    return np.hstack(np.vsplit(_big_vector, self.ts_s))

def group_components(self, r, return_df=False):
    '''Compute the sum of first r chosen components from decomposed series
    (reconstruction procedure).

    :param r: The number of components for series reconstruction.

    :param return_df: If True, then return the pandas.DataFrame with
    reconstructed series.

    :return: pandas.DataFrame with reconstructed series.'''
    self.r = r
    self.C = self.diag_procedure()
    C_grouped = np.hsplit(self.C, self.ts_s)
    res = []
    for i in range(len(C_grouped)):
        res.append(
            np.sum(C_grouped[i][:, :r], axis=1)
        )

    ### Resids part #####
    resids = []
    for i in range(len(C_grouped)):
        resids.append(
            np.sum(C_grouped[i][:, r:], axis=1)
        )
    self.resids = np.matrix(resids)
    #####

    self.C_grouped = np.matrix(res)
    if return_df == True:
        return pd.DataFrame(res, index=['Grouped_component_' + str(i) for i
in self.ts_name]).T

def L_reccurent_forecast(self, steps_ahead):
    '''Compute the recurrent forecast based on columns (MSSA-L).

    :param steps_ahead: The length of the forecast (how many steps ahead to
    compute).

    :return: pandas.DataFrame with reconstructed series and their
    forecaasts.'''

    r = self.r
    v_2 = 0
    for i in range(r):
        v_2 += (self.U[-1, i]) ** 2
    R_L_sum = 0
    for i in range(r):
        R_L_sum += self.U[-1, i] * self.U[:-1, i]
    R_L = (1 / (1 - v_2)) * R_L_sum
    C_grp_forc = self.C_grouped
    N = self.ts_N

```

```

        for i in range(steps_ahead):
            Z = C_grp_forc[:, N - self.embedding_dimension + 1:]
            R_N = np.dot(Z, R_L)
            C_grp_forc = np.hstack((C_grp_forc, R_N))
            N += 1

        self.forc = C_grp_forc
        self.reccurent_coef = R_L
        self.steps_ahead = steps_ahead
        return self.C, pd.DataFrame(C_grp_forc, index=['Forecast_' + str(i) for
i in self.ts_name]).T

    @staticmethod
    def _recursive_coef_calc(a, steps):
        a = np.array(sum(a.tolist(), []))
        psy_list = []
        psy = np.zeros((len(a), 1))
        psy[0] = 1
        for i in range(steps):
            psy_j = np.dot(a, psy).prod()
            psy = np.roll(psy, 1)
            psy[0] = psy_j
            psy_list.append(psy_j ** 2)
        return psy_list

    def conf_int(self):
        """
        Build the confidence intervals for all forecasted series
        :return: pandas DataFrame with conf. intervals in the same order as
series in original table
        """
        intervals = []
        out_col_names = []
        for r in range(len(self.resids)):
            mu, std = norm.fit(np.squeeze(np.asarray(self.resids[r])))
            p5 = np.percentile(np.squeeze(np.asarray(self.resids[r])) - mu, 5)
            p95 = np.percentile(np.squeeze(np.asarray(self.resids[r])) - mu, 95)

            recursive_coefs = self._recursive_coef_calc(self.reccurent_coef,
self.steps_ahead)
            recursive_coefs = [1] + recursive_coefs
            forc_interval = []
            for i in range(self.steps_ahead):
                forc_interval.append(
                    sum(recursive_coefs[:i+1])*(std**2)
                )

            upper_bound_in = (np.squeeze(np.array(self.forc[r, :-
self.steps_ahead])) + p95).tolist()
            lower_bound_in = (np.squeeze(np.array(self.forc[r, :-
self.steps_ahead])) + p5).tolist()
            upper_bound_out = (np.squeeze(np.array(self.forc[r, -
self.steps_ahead:])) + np.sqrt(np.array(forc_interval))*1.96).tolist()
            lower_bound_out = (np.squeeze(np.array(self.forc[r, -
self.steps_ahead:])) - np.sqrt(np.array(forc_interval))*1.96).tolist()
            intervals.append(lower_bound_in + lower_bound_out)
            intervals.append(upper_bound_in + upper_bound_out)

```

```
        out_col_names += ["conf. 5%", "conf. 95%"]
    conf_int_df = pd.DataFrame(intervals).T
    conf_int_df.columns = out_col_names
    return conf_int_df
```