Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Казанский (Приволжский) федеральный университет»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ
И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА МАТЕМАТИЧЕСКОЙ СТАТИСТИКИ

Направление: 01.04.02 – Прикладная математика и информатика
Магистерская программа: Методы прикладной математической статистики

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

МЕТОДЫ БОРЬБЫ С НЕУСТОЙЧИВОСТЯМИ КОРРЕЛЯЦИЙ ВРЕМЕННЫХ РЯДОВ

Обучающийся 2 курса
группы 09-815 Фархшатов Ф.Р.

Руководитель
канд. физ.-мат. наук, ассистент Новиков А.А.

Заведующий кафедрой математической статистики
канд. физ.-мат. наук, доцент Симушкин С.В.

Казань – 2020

Содержание

1 Введение

Классической задачей математической статистики является выведение точечных оценок различных параметров. Например, оценки среднего, дисперсии, коэффициентов корреляции. Аналогичная проблема возникает при построении характеристик временных рядов.

Как известно, для измерения степени зависимости между наблюдаемыми величинами можно использовать коэффициент корреляции. Однако, в процессах, протекающих во времени, зависимости между величинами могут проявляется не мгновенно, а с некоторым запаздыванием (временным лагом). В этом случае, оценка зависимости путем вычисления коэффициента корреляции может давать результат не только неточный, но и совершенно неверный. Таким образом, возникает задача поиска лагов во временных рядах.

Оценка величины отложенного влияния имеет большое практическое значение как при рассмотрении стационарных временных рядов (например, физические процессы, протекающие в промышленных системах [1]), так и динамических систем (финансовые рынки [2], рынки деривативов [3]). Также встречаются применения в анализе и обработке сигналов: радиолокация, сейсмология, гидролокация, геофизика и т.д. ([4], [5], [6]).

Целью работы является сравнительный анализ методов поиска временного лага на основе методов Монте-Карло.

В соответствии с поставленной целью были сформулированы следующие задачи:

1. Сгенерировать совокупность пар временных рядов с различными временными лагами и функциональными зависимостями между собой.

2. Программно реализовать методы максимума корреляций и минимальной дисперсии корреляций.

3. Проверить эффективность указанных методов для сгенерированного набора данных.

2 Предварительные сведения и обозначения

Напомним понятие временного ряда (time series). Согласно [7] и [8], *временной ряд* - это последовательность числовых показателей, упорядоченных во времени. В данной работе будет представлен в виде вектора $\mathbf{x} = (x_1, ..., x_n)$, с числовыми элементами $x_1, ..., x_n \in \mathbb{R}$, записанными через равные промежутки времени.

Срезом (slice) временного ряда $\mathbf{x}$ от i-го до j-го момента времени будем называть временной ряд $\mathbf{x}[i:j] := (x_i, ..., x_j)$.

2.1 Метод максимумов корреляций

Опишем существующий метод поиска временного лага: *метод максимумов корреляций.* Данный подход (generalized crosscorrelation method) был описан в работах [4] и [5]. А также в работах [6], [9], [10] он используется для поиска лага между сильно скоррелированными аудио сигналами.

Пусть $\mathbf{x}, \mathbf{y}$ - временные ряды длинны n. Метод заключается в поиске такого лага s, при сдвиге на который ($\mathbf{y}$ относительно $\mathbf{x}$) коэффициент корреляции максимален. То есть, решается следующая задача оптимизации:

$$cor\Big(\mathbf{x}[1:n-s],\ \mathbf{y}[s:n]\Big) \underset{s}{\to} max, \tag{1}$$

$$s^* = argmax\left(cor\Big(\mathbf{x}[1:n-s],\ \mathbf{y}[s:n]\Big)\right). \tag{2}$$

2.2 Метод минимальной дисперсии корреляций

Пусть $\mathbf{x}, \mathbf{y}$ - временные ряды длинны n. Через $Cor^s(\mathbf{x}, \mathbf{y})$ обозначим вектор, составленный из значений коэффициента корреляции срезов $\mathbf{x}$ и $\mathbf{y}$ с лагом s, то есть:

$$Cor^s(\mathbf{x}, \mathbf{y}) = \Big(cor(\mathbf{x}[k:k+h], \mathbf{y}[k+s:k+s+h])\Big)_{k=1}^{n-h},$$

где h - фиксированный размер срезов.

Метод состоит в поиске такого временного лага s, при котором дисперсия коэффициентов корреляции $Var(Cor^s(\mathbf{x}, \mathbf{y}))$ минимальна. Таким образом получаем следующую задачу оптимизации:

$$Var(Cor^s(\mathbf{x}, \mathbf{y})) \underset{s}{\to} min, \tag{3}$$

$$s^* = argmin\Big(Var(Cor^s(\mathbf{x}, \mathbf{y}))\Big). \tag{4}$$

Проиллюстрируем введенный метод на примере. Сгенерируем две линейно зависимые нормально распределенные выборки $\mathbf{x}$ и $\mathbf{y}$, затем искусственно сдвинем их друг относительно друга с лагом равным 21. После этого применим предложенный метод для отыскания временного лага. На рисунках 1, 2 изображены графики коэффициентов корреляции $Cor^s(\mathbf{x}, \mathbf{y})$ для $\mathbf{x}$ и $\mathbf{y}$. На рисунке 1 - без учета временного лага, а на рисунке 2 - с учетом лага, найденного при помощи данного метода. Исходя из расчетных значений (см. таблицу 1), можно сделать вывод, что величина лага была найдена корректно. Реализация данного примера приведена в Приложении.

Параметр	Истинные значения	Без учета лага	После применения метода
Корреляция	0.8327	0.0257	0.8327
Лаг	21	0	21
Дисперсия	0.0013	0.0629	0.0013

Таблица 1: Результаты применения метода

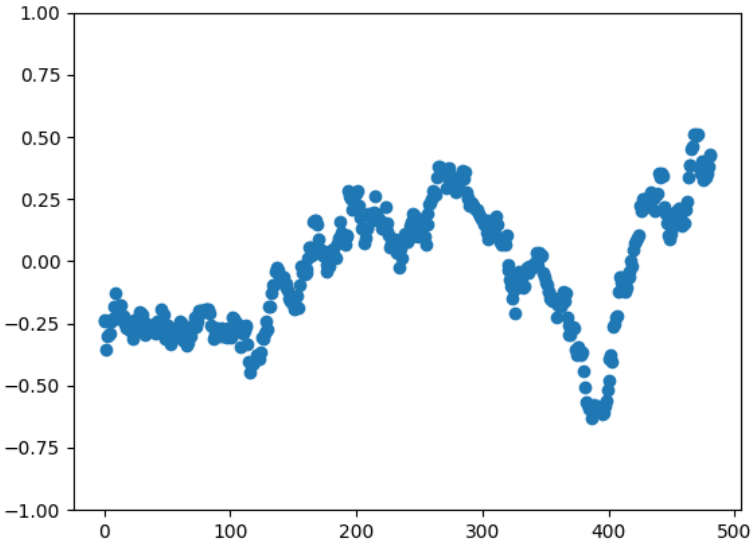

Рис. 1: $Cor^s(\mathbf{x}, \mathbf{y})$ для $s = 0$

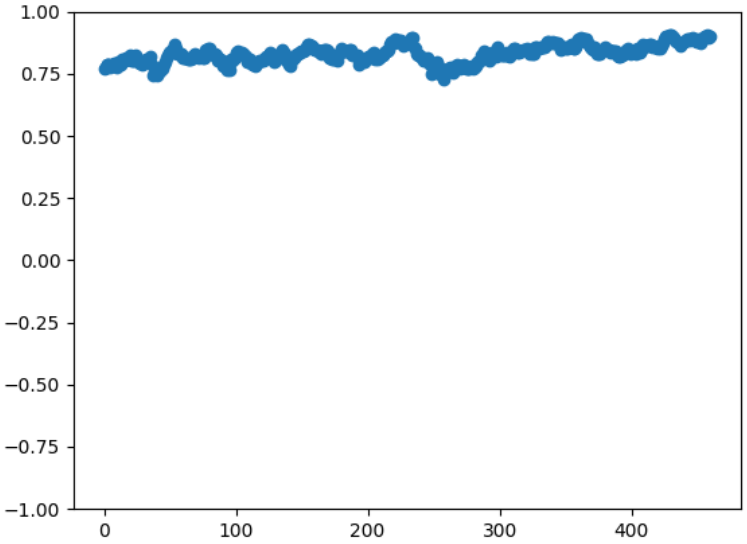

Рис. 2: $Cor^s(\mathbf{x}, \mathbf{y})$ для $s = 21$

2.3 Метод главных компонент

Метод главных компонент (principal component analysis) позволяет уменьшить размерность изучаемых данных, причем так, чтобы количество потерянной информации было минимальным.

В одном из классических подходов [11], метод главных компонент сводится к вычислению собственных векторов и собственных значений корреляционной матрицы исходных данных. Опишем данный подход.

Пусть $X = (\mathbf{x}^1, ..., \mathbf{x}^p)$ – p-мерный вектор, с вектором средних $\mu = (\mu_1, ..., \mu_p)$. Тогда вектор $\zeta = (\zeta_1, \ldots, \zeta_p)'$ называется *вектором главных компонент*, если

1. Все компоненты вектора ζ являются линейными комбинациями вектора X
2. Дисперсия первой компоненты вектора ζ_1 максимальна среди всех нормированных линейных комбинаций X
3. Дисперсия j-ой компоненты вектора ζ_j максимальна среди всех нормированных линейных комбинаций X, не коррелирующих с $\zeta_1, \ldots, \zeta_{j-1}, j = 1, \ldots, p$

Пусть $\mathbf{C}$ – матрица корреляций X. Тогда определяющие главные компоненты вектор-столбцы v_j совпадают с нормированными собственными векторами матрицы $\mathbf{C}$ то есть:

$$\mathbf{C}v_j = v_j, \quad j = 1, \ldots, p$$

причем собственные числа выбраны так, что $\gamma_1 \geqslant \cdots \geqslant \gamma_p$, а все собственные векторы ортогональны.

Для выяснения количества объясняющих переменных известна следующая рекомендация ([11]). Рассмотрим следующие величины:

$$m_q = \frac{\gamma_1 + \cdots + \gamma_q}{\gamma_1 + \cdots + \gamma_p}, \tag{5}$$

$$n_q = m_q \cdot 100\%. \tag{6}$$

Величина n_q показывает процент наблюдений, которые могут быть объяснены факторами, определяющими первые q главных компонент. Если n_q достаточно велико ($>75\%$), то все остальные $p - q$ компонент можно не рассматривать. В таком случае q – это количество объясняющих переменных.

3 Постановка вычислительных экспериментов

3.1 Сравнение методов

Опишем постановку вычислительного эксперимента, проведенного с целью сравнения метода максимума корреляции (1), (2) с методом минимума дисперсии корреляций (3), (4).

Проведем N испытаний, каждое из которых состоит из следующей последовательности действий:

1. Генерирование временных рядов $\mathbf{x}$ и $\mathbf{y}$ длины n с величиной лага s.

2. Вычисление оценки временного лага с использованием метода максимума корреляций:
$$s^* = argmax\Big(cor\Big(\mathbf{x}[1:n-s],\ \mathbf{y}[s:n]\Big)\Big).$$

3. Вычисление оценки временного лага с использованием метода минимума дисперии корреляций:
$$\hat{s} = argmin\Big(Var(Cor^s(\mathbf{x},\mathbf{y}))\Big).$$

Таким образом, мы получаем следующие числовые наборы:

1. $\mathbf{s} = (s_1, ..., s_N)$ - набор с истинными значениями лагов.

2. $\mathbf{s}^* = (s_1^*, ..., s_N^*)$ - набор с оценками лагов, вычисленных при помощи метода максимума корреляций.

3. $\hat{\mathbf{s}} = (\hat{s}_1, ..., \hat{s}_N)$ - набор с оценками лагов, вычисленных методом минимальной дисперсии корреляций.

Опишем более подробно способы генерирования временных рядов $\mathbf{x}$ и $\mathbf{y}$. Для генерирования $\mathbf{x} = (x_i)_{i=1}^n$ используются две следующие модели:

1. Модель стационарного временного ряда, распределенного нормально со средним μ и дисперсией σ^2:
$$\mathbf{x} \sim \mathcal{N}(\mu, \sigma^2).$$

2. Модель случайного блуждания
$$x_1 = d + w_i,$$

$$x_i = x_{i-1} + w_i,$$

с нормально распределенным шагом $\mathbf{w} = (w_i)_{i=1}^n \sim \mathcal{N}(0, \sigma^2)$ и начальной точкой d.

Ряд $\mathbf{y} = (y_i)_{i=1}^n$ генерируется на основе $\mathbf{x} = (x_i)_{i=1}^n$, следующим образом:

$$y_i = f(x_i) + \varepsilon_i,$$

$$\varepsilon = (\varepsilon_i)_{i=1}^n \sim \mathcal{N}(0, \sigma^2),$$

где f - некоторая функциональная зависимость, а ε - сгенерированный шум.

Таким образом, для каждой модели временных рядов $\mathbf{x}$, $\mathbf{y}$ и для каждой изучаемой функциональной зависимости f проводится вычислительный эксперимент, состоящий из N испытаний. В ходе эксперимента определяется эффективность методов оценки временного лага (1), (2) и (3), (4) на основе вычисления некоторых функций потерь между истинными значениями лага $\mathbf{s} = (s_1, ..., s_N)$ и оценками $\mathbf{s}^* = (s_1^*, ..., s_N^*)$ и $\hat{\mathbf{s}} = (\hat{s}_1, ..., \hat{s}_N)$.

В качестве функции потерь будем использовать среднюю ошибку (Mean Error - ME), среднюю абсолютную ошибку (Mean Absolute Error - MAE) и квадратный корень из среднеквадратической ошибки (Root Mean Square Error - RMSE):

$$ME(\bar{\mathbf{s}}) = \frac{1}{N}\sum_{i=1}^{N}(s_i - \bar{s}_i),$$

$$MAE(\bar{\mathbf{s}}) = \frac{1}{N}\sum_{i=1}^{N}|s_i - \bar{s}_i|,$$

$$RMSE(\bar{\mathbf{s}}) = sd(\mathbf{s} - \bar{\mathbf{s}}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(s_i - \bar{s}_i)}.$$

3.2 Поиск внутренней размерности

Опишем постановку вычислительного эксперимента, проведенного с целью сравнения эффективности снижения размерности методом главных компонент. Применительно к случаям, когда матрица корреляции изучаемого набора данных вычислена

1. без учета временных лагов,
2. с учетом лагов, в соответствии с методом максимума корреляции (1), (2),
3. с учетом лагов, в соответствии с методом минимума дисперсии (3), (4).

Проведем N испытаний, каждое из которых состоит из следующей последовательности действий:

1. Генерирование набора данных $X = (\mathbf{x}^1, ..., \mathbf{x}^p)$.

2. Вычисление матрицы корреляций $C = (c_{ij})$, где $c_{ij} = cor(\mathbf{x}^i, \mathbf{x}^j)$.

3. Вычисление матрицы корреляций $C = (c_{ij})$ в соответствии с методом максимума корреляции (1), (2):

$$c_{ij} = max_s \left(cor\Big(\mathbf{x}^i[1 : n - s],\ \mathbf{x}^j[s : n]\Big) \right).$$

4. Вычисление матрицы корреляций $C = (c_{ij})$ в соответствии с методом минимума дисперсии корреляций (3), (4):

$$c_{ij} = cor\Big(\mathbf{x}^i[1 : n - \hat{s}], \mathbf{x}^j[\hat{s} : n]\Big),$$

$$\text{где} \quad \hat{s} = argmin\Big(Var(Cor^s(\mathbf{x}, \mathbf{y}))\Big).$$

5. Применение метода главных компонент к каждому из случаев вычисления матрицы корреляций. Вычисление показателя n_q в соответствии с (5), (6).

Набор данных $X = (\mathbf{x}^1, ..., \mathbf{x}^p)$ генерируется таким образом, чтобы его истинное количество объясняющих переменных было равно некоторому q. Генерируются q стационарных нормально распределенных временных рядов

$$\mathbf{u}^1, ..., \mathbf{u}^q \sim \mathcal{N}(\mu, \sigma^2).$$

Каждый из векторов $\mathbf{x}^1, ..., \mathbf{x}^p$ является линейной комбинацией переменных $\mathbf{u}^1, ..., \mathbf{u}^q$ с некоторыми случайными коэффициентами $\alpha_1, ..., \alpha_q$ и случайными временными лагами $s_1, ..., s_q$:

$$\mathbf{x}^i = \sum_{k=1}^{q} \alpha_k \mathbf{u}^k[s_k : n + s_k], \qquad \text{для} \quad i = 1, ..., p.$$

В результате эксперимента, перечисленные подходы вычисления матрицы корреляций сравниваются при помощи показателей n_q. А именно: для каждого метода берется процент таких испытаний, в которых $n_q > 75\%$.

4 Результаты

4.1 Сравнение методов

В таблицах (2), (3) предоставлены результаты вычислительного эксперимента по сравнению эффективности оценки временного лага методами максимума корреляций (1), (2) и минимума дисперсии корреляций (3), (4).

	Максимум корреляций			Минимум дисперсий		
	MAE	RMSE	ME	MAE	RMSE	ME
$a\mathbf{x} + b$	0.00	0.00	0.00	0.00	0.00	0.00
$a\mathbf{x}^2 + b\mathbf{x} + c$	0.00	0.00	0.00	0.00	0.00	0.00
$a/\mathbf{x} + b$	9.14	10.83	-0.35	2.20	5.32	-0.15
$a/\mathbf{x}^2 + b$	9.20	10.86	-0.51	4.51	7.67	-0.89
$e^{\mathbf{x}}$	0.00	0.00	0.00	0.19	1.49	-0.07
$e^{-\mathbf{x}}$	9.40	11.07	-0.21	0.05	0.65	0.00
$e^{-\mathbf{x}^2}$	9.10	10.78	0.06	6.20	8.86	-0.76
$\sin(\mathbf{x})$	9.48	11.20	-0.25	1.67	4.58	-0.20
$\cos(\mathbf{x})$	0.00	0.00	0.00	9.11	10.80	-0.74

Таблица 2: Результаты вычислительного эксперимента для модели стационарного временного ряда.

Сначала рассмотрим стационарный случай. Из таблицы (2) видно, что в случаях линейной и параболической зависимости оба метода работают отлично т.е. найденные при помощи методов лаги полностью совпадает с истинными лагами. Метод минимума дисперсии корреляций дает более точную оценку в случаях $a/\mathbf{x} + b$, $e^{-\mathbf{x}}$ и $\sin(\mathbf{x})$, а в случаях $e^{\mathbf{x}}$ и $\cos(\mathbf{x})$, наоборот, более точную оценку дает метод максимума корреляции. Особенно интересными представляются случаи зависимостей $\sin(\mathbf{x})$ и $\cos(\mathbf{x})$, поскольку методы показывают диаметрально противоположные результаты. Аналогично и в случаях $e^{\mathbf{x}}$ и $e^{-\mathbf{x}}$.

В рамках модели случайного блуждания (см. таблицу (3)), метод минимума дисперсий по-прежнему дает лучший результат для зависимостей $a/\mathbf{x} + b$, $e^{-\mathbf{x}}$, а метод максимума корреляций для $e^{\mathbf{x}}$. Отметим, что в случае нестационарных временных рядов, оба метода в целом показывают более низкий результат, чем в случае стационарных временных рядов.

	Максимум корреляций			Минимум дисперсий		
	MAE	RMSE	ME	MAE	RMSE	ME
$a\mathbf{x} + b$	0.00	0.00	0.00	0.00	0.03	0.00
$a\mathbf{x}^2 + b\mathbf{x} + c$	0.99	4.02	0.99	1.99	5.33	-0.39
$a/\mathbf{x} + b$	17.54	18.44	-1.14	3.04	6.41	-0.14
$a/\mathbf{x}^2 + b$	17.54	18.38	-1.75	3.47	6.50	-0.12
$e^{\mathbf{x}}$	1.44	4.49	0.53	8.80	10.41	-1.65
$e^{-\mathbf{x}}$	17.65	18.27	-0.70	8.97	10.63	-1.02
$e^{-\mathbf{x}^2}$	14.33	15.64	-2.19	10.03	11.39	-0.32
$\sin(\mathbf{x})$	10.66	13.17	-0.03	12.46	13.31	-1.00
$\cos(\mathbf{x})$	10.71	13.28	0.26	12.34	13.12	-1.02

Таблица 3: Результаты вычислительного эксперимента для модели случайного блуждания.

4.2 Поиск внутренней размерности

В таблице (4) приведены результаты вычислительного эксперимента снижения размерности методом главных компонент для следующих параметров: $q = 2,\ p = 3,\ N = 10000$.

Метод	Без учета лага	Минимум дисперсий	Максимум корреляций
%	27.32	63.59	100

Таблица 4: Процент правильного обнаружения количества объясняющих переменных.

Из таблицы (4) следует, что расчет матрицы корреляций с учетом временного лага с помощью метода минимума дисперсий лучше помогает выяснить количество объясняющих переменных, чем без учета лага. Однако, метод максиму корреляций дает наилучший результат.

5 Заключение

В работе рассмотрены методы оценки отложенного влияния во временных рядах с различными видами функциональной зависимости. Исследовав методы максимума корреляций и минимальной дисперсии корреляций, целесообразно сделать вывод о том, что в целом методы не равны, то есть, в ряде случаев методы сходятся к различным значениям (см. таблицы 2, 3). Кроме того в работе рассмотрены случаи, в которых методы сходятся к одному значению.

Из чего следует, что иногда методы перекрывают друг друга. Поэтому рекомендуется использовать оба метода для получения более достоверной оценки лага по времени.

Отметим, что метод минимальной дисперсии корреляций введен в настоящей работе впервые. Следует также упомянуть, что вычислительная сложность метода минимальной дисперсии корреляций является существенно большей, чем у метода максимумов корреляций.

Список литературы

[1] Robert L. Mason and John C. Young. *Multivariate Statistical Process Control with Industrial Application*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 2001.

[2] Xingyu Zhou, Zhisong Pan, Guyu Hu, Siqi Tang, and Cheng Zhao. Stock market prediction on high-frequency data using generative adversarial nets. *Mathematical Problems in Engineering*, 2018, 04 2018. https://doi.org/10.1155/2018/4907423.

[3] Chorro Christophe, Dominique Guegan, Dominique Guegan, and F. Lelpo. *A time series approach to option pricing: Models, Methods and Empirical Performances*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2015. http://dx.doi.org/10.1007/978-3-662-45037-6.

[4] B. H. V. S. Narayana Murthy, B. Yegnanarayana, and Sudarsana Reddy Kadiri. Time delay estimation from mixed multispeaker speech signals using single frequency filtering. *Circuits, Systems, and Signal Processing*, Volume 39, 08 2019. https://doi.org/10.1007/s00034-019-01239-2.

[5] Jingdong Chen, Jacob Benesty, and Yiteng Huang. Time delay estimation in room acoustic environments: An overview. *EURASIP J. Appl. Signal Proces*, 26503, 01 2006. https://www.researchgate.net/publication/27355862.

[6] Matthew Brandon Rhudy. Real time implementation of a military impulse classifier. *University of Pittsburgh, Master's Thesis*, 2009. http://d-scholarship.pitt.edu/9773/.

[7] Юзбашев М.М. Афанасьев В.Н. *Анализ временных рядов и прогнозирование: Учебник*. М.: Финансы и статистика, Москва, 2001.

[8] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics, Hoboken, New Jersey, 2015.

[9] Jeffrey Vipperman, Matthew Rhudy, and Brian Bucci. Development and implementation of metrics for identifying military impulse noise. *University of Pittsburgh*, page 70, 09 2010. https://www.researchgate.net/publication/235145005.

[10] Matthew Rhudy, Brian Bucci, Jeffrey Vipperman, Jeffrey Allanach, and Bruce Abraham. Microphone Array Analysis Methods Using Cross-Correlations. *ASME International Mechanical Engineering Congress and Exposition*, Volume 15: Sound, Vibration and Design:281–288, 11 2009. https://doi.org/10.1115/IMECE2009-10798.

[11] Симушкин С.В. *Многомерный статистический анализ. Часть 1.* Казанский государственный университет, Казань, 2006.

Приложение

Исходный код размещен в следующем репозитории:

`https://github.com/tosinabase/time-series-delay-estimation`

Все вычисления произведены на языке Python. Структура проекта выглядит следующим образом:

```
project/
    compare_methods.py
    generation.py
    generate_and_check.py
    generate_and_check_parallel.py
    merge_files.py
    dimension_reduction_results.py
    visualize_cors_for_two_vars.py
    efficiency_comparison/
        __init__.py
        check.py
        generator.py
        utils.py
    lag_searching_methods/
        __init__.py
        correlations_maximum.py
        variance_minimum.py
        utils.py
```

project/compare_methods.py

```
import numpy as np
import pandas as pd
from time import time

from efficiency_comparison import Checker, Generator


def generate(gen):
    gen.generate_linear_case(a=2, c=5)
    gen.generate_parabolic_case(a=15, b=0, c=4)
    gen.generate_hyperbolic_case(a=1, c=4)

    gen.generate_by_function(lambda x: 1 / x ** 2, '1/x^2')
    gen.generate_by_function(np.exp, 'e^x')
    gen.generate_by_function(lambda x: np.exp(-x), 'e^(-x)')
    gen.generate_by_function(lambda x: np.exp(-x ** 2), 'e^(-x^2)')
    gen.generate_by_function(np.sin, 'sin')
    gen.generate_by_function(np.cos, 'cos')


number = 1000
gen_rw = Generator(total_num=number, noise_part=0.1,
                   mode='random-walk',
                   start_point=10., std=0.5)
gen_st = Generator(total_num=number, noise_part=0.1,
                   mean=3.5, std=1,
                   mode='stationary')

generate(gen_rw)
generate(gen_st)

```

```
ch_rw = Checker(gen_rw)
start = time()
dfs_rw, report_rw = ch_rw.check_all()
print(time() - start)

ch_st = Checker(gen_st)
start = time()
dfs_st, report_st = ch_st.check_all()
print(time() - start)
```

project/generation.py

```
import numpy as np


def generate_data(n_row, n_col,
                  n_prin_var,
                  mean_prin_var=0,
                  sd_prin_var=1,
                  lag_interval=(5, 15),
                  coefs_interval=(1, 2)):

    # Генерирует набор данных в виде питру матрицы.
    # n_row - количество строк полученных данных,
    # n_col - количество столбцов (переменных),
    # n_prin_var - количество объясняющих переменных.

    n_row_prin_var = n_row + lag_interval[1]
    principal_variables = np.zeros((n_row_prin_var, n_prin_var))
    for i in range(0, n_prin_var):
        principal_variables[:, i] = np.random.normal(mean_prin_var,
        ↪  sd_prin_var, n_row_prin_var)

    coefs = np.random.uniform(coefs_interval[0], coefs_interval[1],
    ↪  (n_col, n_prin_var))
```

```
    lags = np.random.randint(lag_interval[0], lag_interval[1], (n_col,
    ↪  n_prin_var))
    data = np.zeros((n_row, n_col))

    for i in range(0, n_col):
        new_var = np.zeros(n_row)
        for j in range(0, n_prin_var):
            start = lag_interval[1] - lags[i, j]
            new_var += coefs[i, j] *
            ↪  principal_variables[start:start+n_row, j]
        data[:, i] = new_var

    return data
```

project/generate_and_check.py

```
import sys
import numpy as np
import pandas as pd
from generation import generate_data
from lag_searching_methods import corrmatrix_var_min, corrmatrix_cor_max

filenum = int(sys.argv[1])

n_row = 300
n_col = 3
n_prin_var = 2

mean_prin_var = 0
sd_prin_var = 1
lag_interval = (5, 25)
coefs_interval = (1, 2)

h = 40
max_lag = 30
```

```

data = generate_data(n_row, n_col, n_prin_var,
                     mean_prin_var, sd_prin_var,
                     lag_interval, coefs_interval)
default_corrmatrix = np.corrcoef(data, rowvar=False)
sh_corrmatrix = corrmatrix_var_min(data, h, max_lag)
max_corrmatrix = corrmatrix_cor_max(data, max_lag)

def_svd = np.linalg.svd(default_corrmatrix)
sh_svd = np.linalg.svd(sh_corrmatrix)
max_svd = np.linalg.svd(max_corrmatrix)

def_df = pd.DataFrame(def_svd[1])
def_df.to_csv("results/default/singular_values_{}.csv".format(filenum),
 ↪  index=None, header=None)
sh_df = pd.DataFrame(sh_svd[1])
sh_df.to_csv("results/shifted/singular_values_{}.csv".format(filenum),
 ↪  index=None, header=None)
max_df = pd.DataFrame(max_svd[1])
max_df.to_csv("results/max_cor/singular_values_{}.csv".format(filenum),
 ↪  index=None, header=None)
```

project/generate_and_check_parallel.py

```
import os
from subprocess import Popen, PIPE

if not os.path.exists('results'):
    os.makedirs('results')

if not os.path.exists('results/default'):
    os.makedirs('results/default')

if not os.path.exists('results/shifted'):
    os.makedirs('results/shifted')
```

```
12
13  if not os.path.exists('results/max_cor'):
14      os.makedirs('results/max_cor')
15
16  number_of_threads = 12
17  start_num = 0
18  end_num = 10000
19
20  # number_of_threads - количество параллельно запущенных скриптов.
21  # start_num, end_num - начальный и последний номера выходных файлов.
22  # Общее количество получаемых файлов должно превышать число потоков.
23
24  processes = []
25  current_num = start_num + number_of_threads
26
27  for num in range(start_num, current_num):
28      p = Popen(["venv/bin/python3", "generate_and_check.py", str(num)],
29                stdout=PIPE, stderr=PIPE)
30      processes.append(p)
31
32
33  while current_num < end_num:
34
35      for i in range(0, number_of_threads):
36
37          if processes[i].poll() is not None:
38
39              # Проверка на наличие ошибок
40              process_output = processes[i].communicate()
41              if process_output[1] != b'':
42                  print('Error on {}:'.format(current_num))
43                  print(process_output[1].decode('utf-8'))
44
```

```
            # Уничтожение завершенного процесса и создание нового
            processes[i].kill()
            processes[i] = Popen(["venv/bin/python3",
             ↪  "generate_and_check.py", str(current_num)],
                                 stdout=PIPE, stderr=PIPE)
            current_num += 1
        else:
            ert = 'ert'

# Будем ждать завершения всех оставшихся процессов
while len(processes) != 0:
    for process in processes:
        if process.poll() is not None:
            process_output = process.communicate()

            if process_output[1] != b'':

                print('Error on {}:'.format(current_num))
                print(process_output[1].decode('utf-8'))

            processes.remove(process)
```

project/merge_files.py

```
import numpy as np
import pandas as pd

default_df = pd.DataFrame(columns=['s0', 's1', 's2'])
shifted_df = pd.DataFrame(columns=['s0', 's1', 's2'])
max_cor_df = pd.DataFrame(columns=['s0', 's1', 's2'])

for i in range(0, 10000):
    default_row =
     ↪  pd.read_csv("results/default/singular_values_{}.csv".format(i),
     ↪  header=None)
```

```
    default_row = default_row.transpose()
    default_row.rename(columns={0: 's0', 1: 's1', 2: 's2'}, index={0: i},
    ↪  inplace=True)
    default_df = pd.concat([default_df, default_row], sort=False)

    shifted_row =
    ↪  pd.read_csv("results/shifted/singular_values_{}.csv".format(i),
    ↪  header=None)
    shifted_row = shifted_row.transpose()
    shifted_row.rename(columns={0: 's0', 1: 's1', 2: 's2'}, index={0: i},
    ↪  inplace=True)
    shifted_df = pd.concat([shifted_df, shifted_row], sort=False)

    max_cor_row =
    ↪  pd.read_csv("results/max_cor/singular_values_{}.csv".format(i),
    ↪  header=None)
    max_cor_row = max_cor_row.transpose()
    max_cor_row.rename(columns={0: 's0', 1: 's1', 2: 's2'}, index={0: i},
    ↪  inplace=True)
    max_cor_df = pd.concat([max_cor_df, max_cor_row], sort=False)

default_df.to_csv("results/default_df.csv", index=None)
shifted_df.to_csv("results/shifted_df.csv", index=None)
max_cor_df.to_csv("results/max_cor_df.csv", index=None)


deflt = pd.DataFrame()
shft = pd.DataFrame()
max_cor = pd.DataFrame()

deflt['x'] = default_df['s0']/default_df['s1']
deflt['y'] = default_df['s1']/default_df['s2']

```

```
shft['x'] = shifted_df['s0']/shifted_df['s1']
shft['y'] = shifted_df['s1']/shifted_df['s2']

max_cor['x'] = max_cor_df['s0']/max_cor_df['s1']
max_cor['y'] = max_cor_df['s1']/max_cor_df['s2']

deflt.to_csv("results/default_relations.csv", index=None)
shft.to_csv("results/shifted_relations.csv", index=None)
max_cor.to_csv("results/max_cor_relations.csv", index=None)
```

project/dimension_reduction_results.py

```
import pandas as pd


def calc_mq(df):
    return (df['s0'] + df['s1']) / (df['s0'] + df['s1'] + df['s2'])


default_df = pd.read_csv("results/default_df.csv")
shifted_df = pd.read_csv("results/shifted_df.csv")
max_cor_df = pd.read_csv("results/max_cor_df.csv")

default_df['ma'] = calc_mq(default_df)
shifted_df['ma'] = calc_mq(shifted_df)
max_cor_df['ma'] = calc_mq(max_cor_df)

def_percentage = (default_df['ma'] >= 0.75).sum() / len(default_df) * 100
min_var_percentage = (shifted_df['ma'] >= 0.75).sum() / len(shifted_df) *
 ↪  100
max_cor_percentage = (max_cor_df['ma'] >= 0.75).sum() / len(max_cor_df) *
 ↪  100

report = f'''
results:
```

```
default corrmatrix = {def_percentage}
min var corrmatrix = {min_var_percentage}
max cor corrmatrix = {max_cor_percentage}
'''

print(report)
```

project/visualize_cors_for_two_vars.py

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.random.mtrand import _rand as global_randstate

from lag_searching_methods import slice_corcoefs,
↪  lag_with_variance_minimum, corcoef_with_lag


output_folder = 'results/'
input_lag = 21
global_randstate.seed(965)


a = np.random.normal(0, 1, 500)
b = 3*a + 2 * np.random.normal(0, 1, 500)
cor_default = np.corrcoef(a, b)[0, 1]

ab_cors = slice_corcoefs(a, b, 40, 0)
plt.scatter(range(len(ab_cors)), ab_cors)
plt.ylim(-1, 1)
plt.savefig(output_folder + 'default_cors.png')
plt.close()

x = np.concatenate((a, np.random.uniform(0, 5, input_lag)))
y = np.concatenate((np.random.uniform(0, 5, input_lag), b))
cor_shifted = np.corrcoef(x, y)[0, 1]
```

```
26
27  xy_cors = slice_corcoefs(x, y, 40, 0)
28  plt.scatter(range(len(xy_cors)), xy_cors)
29  plt.ylim(-1, 1)
30  plt.savefig(output_folder + 'cors_shifted_vars.png')
31  plt.close()
32
33  lag, rev_ord = lag_with_variance_minimum(x, y, 40, 30)
34  cor_min_var = corcoef_with_lag(x, y, lag, rev_ord)
35
36  xy_cors_min_var = slice_corcoefs(x, y, 40, lag)
37  plt.scatter(range(len(xy_cors_min_var)), xy_cors_min_var)
38  plt.ylim(-1, 1)
39  plt.savefig(output_folder + 'cors_with_optimal_lag.png')
40  plt.close()
41
42  params = {'input_lag': input_lag,
43            'true_cor': cor_default,
44            'true_var': np.var(ab_cors),
45            'false_cor': cor_shifted,
46            'false_lag': 0,
47            'false_var': np.var(xy_cors),
48            'founded_cor': cor_min_var,
49            'founded_lag': lag,
50            'founded_var': np.var(xy_cors_min_var)}
51
52  report = '''
53  Истинный Лаг = {input_lag}
54
55  Истинные показатели:
56  Коэффициент корреляции = {true_cor}
57  Лаг, с которым посчитан коэффициент корреляции = 0
58  Дисперсия = {true_var}
```

```
59
60  До нахождения оптимального лага:
61  Коэффициент корреляции = {false_cor}
62  Лаг, с которым посчитан коэффициент корреляции = {false_lag}
63  Дисперсия = {false_var}
64
65  После нахождения оптимального лага:
66  Коэффициент корреляции = {founded_cor}
67  Лаг, с которым посчитан коэффициент корреляции = {founded_lag}
68  Дисперсия = {founded_var}
69  '''
70
71  report = report.format(**params)
72  tex_report = r'''
73      \hline
74      Параметр & Истинные значения & Без учета лага & После применения
    ↪   метода \\ \hline
75      Корреляция & {true_cor:.4f} & {false_cor:.4f} & {founded_cor:.4f} \\
    ↪   \hline
76      Лаг & {input_lag} & 0 & {founded_lag}   \\ \hline
77      Дисперсия &  {true_var:.4f} & {false_var:.4f} & {founded_var:.4f} \\
78      \hline
79  '''
80
81  tex_report = tex_report.format(**params)
82
83  print(report)
84  print(tex_report)
```

project/efficiency_comparison/__init__.py

```
1  from .generator import Generator
2  from .check import Checker
```

```
from .utils import linear_function, parabolic_function,
↪ hyperbolic_function
```

project/efficiency_comparison/check.py

```
import json
import os
import numpy as np
import pandas as pd
from multiprocessing import Pool
from sklearn.metrics import mean_squared_error, mean_absolute_error

from efficiency_comparison import Generator
from lag_searching_methods import \
    lag_var_min_current_order, lag_and_coef_with_cor_max


class Checker:
    generator = None
    max_lag = None
    h = None
    max_cor_result = None
    min_var_result = None
    results = None
    report = None

    def __init__(self, generator, max_lag=30, h=40):
        self.generator = generator
        self.max_lag = max_lag
        self.h = h

    def max_cor_for_array(self, data):
        result = [lag_and_coef_with_cor_max(x, y, self.max_lag)[0] for x,
        ↪ y in data]
        return result
```

```
30
31      def var_min_for_array(self, data):
32          result = [lag_var_min_current_order(x, y, self.h, self.max_lag)
            ↪  for x, y in data]
33          return result
34
35      @staticmethod
36      def check_method(data, method):
37          n_parts = os.cpu_count()
38          parts = [part.tolist() for part in np.array_split(data, n_parts)]
39
40          pool = Pool(n_parts)
41          result_parts = pool.map(method, parts)
42
43          result = []
44          for part in result_parts:
45              result.extend(part)
46
47          pool.close()
48
49          return result
50
51      def max_corr_by_data(self, data):
52          result = self.check_method(data, self.max_cor_for_array)
53          return result
54
55      def min_var_by_data(self, data):
56          result = self.check_method(data, self.var_min_for_array)
57          return result
58
59      def check_max_cor(self):
60          self.max_cor_result = {'real': self.generator.real_lags}
61          for name in self.generator.data:
```

```
62              result = self.max_corr_by_data(self.generator.data[name])
63              self.max_cor_result[name] = result
64
65      def check_min_var(self):
66          self.min_var_result = {'real': self.generator.real_lags}
67          for name in self.generator.data:
68              result = self.min_var_by_data(self.generator.data[name])
69              self.min_var_result[name] = result
70
71      def check_all(self):
72          self.check_max_cor()
73          self.check_min_var()
74
75          return self.report_all()
76
77      @staticmethod
78      def make_report(result):
79          df = pd.DataFrame(result)
80          report = {'accuracy': {},
81                    'ME': {},
82                    'RMSE': {},
83                    'MAE': {},
84                    'std': {}}
85
86          for col in df.drop('real', axis=1).columns:
87              report['accuracy'][col] = (df[col] == df['real']).sum() /
                ↪  len(df)
88              report['ME'][col] = (df['real'] - df[col]).mean()
89              report['RMSE'][col] = mean_squared_error(df['real'], df[col],
                ↪  squared=False)
90              report['MAE'][col] = mean_absolute_error(df['real'], df[col])
91              report['std'][col] = (df['real'] - df[col]).std()
92
```

```
        return df, report

    def report_all(self):
        self.report = {}
        self.results = {}
        for result, method_name in [(self.max_cor_result, 'max_cor'),
                                    (self.min_var_result, 'min_var')]:
            df, method_report = self.make_report(result)
            self.results[method_name] = df
            self.report[method_name] = method_report

        print(json.dumps(self.report, indent=10))
        return self.results, self.report


if __name__ == '__main__':
    from time import time
    gen = Generator(total_num=100)
    gen.generate_linear_case()

    ch = Checker(gen)
    print('started')
    start = time()
    ch.check_min_var()
    print(time() - start)

    print('Success')
```

project/efficiency_comparison/generator.py

```
import numpy as np

from efficiency_comparison.utils import linear_function,
↪  parabolic_function, hyperbolic_function

```

```
5
6   class Generator:
7       real_lags = None
8       kernel = None
9
10      total_num = None
11      size = None
12      kernel_mean = None
13      kernel_std = None
14      noise_part = 0.1
15
16      data = None
17
18      def __init__(self, total_num, vector_size=300,
19                   min_lag=5, max_lag=25,
20                   noise_part=0.1,
21                   mode='stationary',
22                   **kwargs):
23          # total_num - количество испытаний,
24          # vector_length - длина каждого вектора,
25          # min_lag - минимальный лаг для генерирования,
26          # max_lag - максимальный лаг для генерирования.
27          # mode = 'stationary' or 'random-walk'
28
29          # mean=0, std=1,
30          # start_point
31          # dist
32
33          self.total_num = total_num
34          self.size = vector_size
35          self.noise_part = noise_part
36          self.data = {}
37
```

```
38          self.real_lags = np.random.randint(min_lag, max_lag, total_num)
39          self.generate_kernel(mode, **kwargs)
40
41      def generate_kernel(self, mode, **kwargs):
42          if mode == 'stationary':
43
44              self.kernel_mean = kwargs.get('mean', 0)
45              self.kernel_std = kwargs.get('std', 1)
46
47              self.kernel = [np.random.normal(self.kernel_mean,
48                                              self.kernel_std,
49                                              lag + self.size)
50                             for lag in self.real_lags]
51
52          elif mode == 'random-walk':
53              start_point = kwargs.get('start_point', 1)
54              distribution = kwargs.get('dist', 'normal')
55              mean = kwargs.get('mean', 0)
56              std = kwargs.get('std', 1)
57              low = kwargs.get('low', -1)
58              high = kwargs.get('high', 1)
59
60              if distribution == 'normal':
61                  self.kernel = [np.concatenate(
62                                      [np.array([start_point]),
63                                       np.random.normal(mean, std, lag +
                                        ↪  self.size)]
64                                              ).cumsum()
65                                 for lag in self.real_lags]
66
67              elif distribution == 'uniform':
68                  self.kernel = [np.concatenate(
69                                      [np.array([start_point]),
```

```
                                    np.random.uniform(low, high, lag +
                                    ↪  self.size)]
                                            ).cumsum()
                                for lag in self.real_lags]

            else:
                raise Exception("Unexpected parameter: distribution. "
                                "Use 'normal or 'uniform'.")

        else:
            raise Exception("Unexpected mode. "
                            "Use 'stationary' or 'random-walk'.")

    def generate_by_function(self, func, name, noise_part=None, **kwargs):
        # y = func(x) + noise
        if noise_part is None:
            noise_part = self.noise_part

        ys = [func(x, **kwargs) for x in self.kernel]

        # add noise
        ys = [y + np.random.normal(0, noise_part * np.std(y), len(y))
              for y in ys]

        data = [(x[lag:], y[:-lag])
                for x, y, lag in zip(self.kernel, ys, self.real_lags)]
        self.data[name] = data

    def generate_linear_case(self, a=1, c=0, noise_part=None):
        # y = a*x + c + noise
        self.generate_by_function(linear_function, 'linear', noise_part,
        ↪  a=a, c=c)
```

```
    def generate_parabolic_case(self, a=1, b=1, c=0, noise_part=None):
        # y = a*x^2 + b*x + c + noise
        self.generate_by_function(parabolic_function, 'parabolic',
        ↪  noise_part, a=a, b=b, c=c)

    def generate_hyperbolic_case(self, a=1, c=0, noise_part=None):
        # y = a/x + c + noise
        self.generate_by_function(hyperbolic_function, 'hyperbolic',
        ↪  noise_part, a=a, c=c)


if __name__ == '__main__':
    gen = Generator(total_num=1000, mode='random-walk', start_point=10.)
    gen.generate_hyperbolic_case(a=1, c=4)
    gen.generate_by_function(np.exp, 'e^x')
    gen.generate_linear_case()
    print('Success')
```

project/efficiency_comparison/utils.py

```
def linear_function(x, a=1, c=0):
    # y = a*x + c
    return a * x + c


def parabolic_function(x, a=1, b=0, c=0):
    # y = a*x^2 + b*x + c
    return a * x**2 + b * x + c


def hyperbolic_function(x, a=1, c=0):
    # y = a / x + c
    return (a / x) + c
```

project/lag_searching_methods/__init__.py

```python
from .correlations_maximum import lag_and_coef_with_cor_max,
    corrmatrix_cor_max
from .variance_minimum import slice_corcoefs, corrmatrix_var_min, \
    corrcoef_with_variance_minimum, lag_var_min_current_order,
        lag_with_variance_minimum
from .utils import corcoef_with_lag
```

project/lag_searching_methods/correlations_maximum.py

```python
import numpy as np


def lag_and_coef_with_cor_max(x, y, max_lag):
    # Возвращает лаг и коэффициент корреляции для массивов x и y,
    # соответствующий максимуму корреляций.

    coefs = [np.corrcoef(x, y)[0, 1]]
    coefs.extend(
        [np.corrcoef(x[:-lag], y[lag:])[0, 1]
         for lag in range(1, max_lag + 1)]
    )

    res_lag = np.argmax(coefs)
    corr = coefs[res_lag]

    return res_lag, corr


def corrmatrix_cor_max(data, max_lag):
    # Вычисляет матрицу корреляций с учетом лагов
    # в соответствии с методом максимумов корреляций
    n_col = data.shape[1]
    corrmatrix = np.zeros((n_col, n_col))
    for i in range(0, n_col):
```

```
        for j in range(0, n_col):
            if i < j:
                x = data[:, i]
                y = data[:, j]
                _, corr = lag_and_coef_with_cor_max(x, y, max_lag)
                _, rev_corr = lag_and_coef_with_cor_max(y, x, max_lag)

                if rev_corr > corr:
                    corr = rev_corr

                corrmatrix[i, j] = corr
                corrmatrix[j, i] = corr

        corrmatrix[i, i] = 1

    return corrmatrix


# Пример:
if __name__ == '__main__':
    input_lag = 21

    a = np.random.normal(0, 1, 300)
    b = 3 * a + 5

    x = np.concatenate((a, [0] * input_lag))
    y = np.concatenate(([0] * input_lag, b))

    lag, cor = lag_and_coef_with_cor_max(x, y, 30)

    report = f'''
    Истинный лаг: {input_lag}
    Лаг, найденный методом максимумов корреляций {lag}
```

```
    '''
    print(report)
```

project/lag_searching_methods/variance_minimum.py

```
import numpy as np


def slice_corcoefs(x, y, h, s):
    # Возвращает массив коэффициентов корреляции,
    # последовательно посчитанных на срезах массивов x и y, где
    # s - велечина сдвига срезов относительно друг друга,
    # h - размер среза.
    res = [np.corrcoef(x[i: i + h], y[i + s: i + h + s])[0, 1] for i in
    ↪  range(0, len(x) - h - s + 1)]
    return res


def lag_var_min_current_order(x, y, h, max_lag):
    # Вычисляет лаг для массивов x и y,
    # соответствующий минимальной дисперсии корреляций.
    # Только в заданном порядке!
    v_min = np.var(slice_corcoefs(x, y, h, 0))
    i = 0
    for lag in range(1, max_lag + 1):
        v = np.var(slice_corcoefs(x, y, h, lag))
        if v < v_min:
            v_min = v
            i = lag

    res_lag = i
    return res_lag


def lag_with_variance_minimum(x, y, h, max_lag):
```

```
    # Вычисляет лаг для массивов x и y,
    # соответствующий минимальной дисперсии корреляций.

    # Возвращает пару (res_lag, reverse_order), где reverse_order имеет
    ↪ булево значение,
    # показывающее порядок массивов x и y, при котором достигается минимум
    ↪ дисперсии.
    # Если reverse_order == True, то это значит, что оптимальный лаг
    ↪ res_lag достигается
    # в обратном порядке массивов x, y.
    v_min = np.var(slice_corcoefs(x, y, h, 0))
    u_min = np.var(slice_corcoefs(y, x, h, 0))
    i = 0
    j = 0
    for lag in range(1, max_lag + 1):
        v = np.var(slice_corcoefs(x, y, h, lag))
        u = np.var(slice_corcoefs(y, x, h, lag))
        if v < v_min:
            v_min = v
            i = lag
        if u < u_min:
            u_min = u
            j = lag
    if v_min < u_min:
        res_lag = i
        reverse_order = False
    else:
        res_lag = j
        reverse_order = True

    return res_lag, reverse_order


```

```
60 def corrcoef_with_variance_minimum(x, y, h, max_lag):
61     # Вычисляет коэффициент корреляции для массивов x и y,
62     # соответствующий минимальной дисперсии корреляций.
63 
64     v_min = np.var(slice_corcoefs(x, y, h, 0))
65     u_min = np.var(slice_corcoefs(y, x, h, 0))
66     i = 0
67     j = 0
68     for lag in range(1, max_lag + 1):
69         v = np.var(slice_corcoefs(x, y, h, lag))
70         u = np.var(slice_corcoefs(y, x, h, lag))
71         if v < v_min:
72             v_min = v
73             i = lag
74         if u < u_min:
75             u_min = u
76             j = lag
77     if v_min <= u_min:
78         coef = np.corrcoef(x[:-i or None], y[i:])[0, 1]
79     else:
80         coef = np.corrcoef(y[:-j or None], x[j:])[0, 1]
81     return coef
82 
83 
84 def corrmatrix_var_min(data, h, max_lag):
85     # Вычисляет матрицу корреляций с учетом лагов
86     # в соответствии с методом минимума дисперсий
87     # коэффициентов корреляции.
88     n_col = data.shape[1]
89     corrmatrix = np.zeros((n_col, n_col))
90     for i in range(0, n_col):
91         for j in range(0, n_col):
92             if i < j:
```

```
                x = data[:, i]
                y = data[:, j]
                corr = corrcoef_with_variance_minimum(x, y, h, max_lag)
                corrmatrix[i, j] = corr
                corrmatrix[j, i] = corr

        corrmatrix[i, i] = 1

    return corrmatrix


# Usage Example:
if __name__ == '__main__':
    a = np.random.normal(0, 1, 300)
    b = a + 2 * np.random.normal(0, 0.5, 300)
    print('Истинное значение корреляции:', np.corrcoef(a, b)[0, 1])

    x = np.concatenate((a, np.random.uniform(0, 5, 21)))
    y = np.concatenate((np.random.uniform(0, 5, 21), b))
    print('Коэффициент корреляции для сдвинутых переменных:',
     ↪  np.corrcoef(x, y)[0, 1])

    lag = lag_var_min_current_order(x, y, 40, 30)
    print(lag)
```

project/lag_searching_methods/utils.py

```
import numpy as np


def corcoef_with_lag(x, y, lag, reverse_order=False):
    # Вычисляет коэффициент корреляции с фиксированным сдвигом, в заданном
     ↪  порядке.

    if reverse_order:
```

```
8           coef = np.corrcoef(y[:-lag or None], x[lag:])[0, 1]
9       else:
10          coef = np.corrcoef(x[:-lag or None], y[lag:])[0, 1]
11
12      return coef
```