МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра математического моделирования и анализа данных

ЦИРУЛЬ Екатерина Алексеевна

СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ СТАТИСТИЧЕСКОГО ОЦЕНИВАНИЯ ПАРАМЕТРОВ МОДЕЛИ АВТОРЕГРЕССИИ ПРИ НАЛИЧИИ ЦЕНЗУРИРОВАНИЯ СПРАВА

Магистерская диссертация
специальность 1-31 80 09 “Прикладная математика и информатика”

Научный руководитель:
Бодягин Игорь Александрович
кандидат физико-математических наук, доцент

Допущена к защите
«__» _____ 2024 г.
Зав. кафедрой математического моделирования и анализа данных
_____ Малюгин Владимир Ильич
подпись
доктор экономических наук, кандидат физико-математических наук, доцент

Минск, 2024

Общая характеристика работы

Магистерская диссертация, 74 страницы, 19 рисунков, 4 таблицы, 16 источников, 4 приложения.

СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ СТАТИСТИЧЕСКОГО ОЦЕНИВАНИЯ ПАРАМЕТРОВ МОДЕЛИ АВТОРЕГРЕССИИ ПРИ НАЛИЧИИ ЦЕНЗУРИРОВАНИЯ СПРАВА

Ключевые слова: МОДЕЛЬ АВТОРЕГРЕССИИ, ЦЕНЗУРИРОВАНИЕ, СТАТИСТИЧЕСКОЕ ОЦЕНИВАНИЕ ПАРАМЕТРОВ, МЕТОД МАКСИМАЛЬНОГО ПРАВДОПОДОБИЯ, СЛУЧАЙНЫЕ ВЕКТОРА.

Объект исследования — модель авторегрессии.

Цель работы — провести сравнительный анализ методов статистического оценивания параметров и выяснить, какие из них являются оптимальными.

Методы исследования — методы теории вероятности и математической статистики, методы иммитационного и статистичского моделирования, методы матричного анализа.

Результаты работы — реализация различных методов статистического оценивания параметров модели авторегрессии, проведённый сравнительный анализ.

Область применения — медицина, экономика, астрономия и другие.

Агульная характарыстыка працы

Магістарская дысертацыя, 74 старонкі, 19 малюнкаў, 4 табліцы, 16 крыніц, 4 дадатка.

ПАРАЎНАЛЬНЫ АНАЛІЗ МЕТАДАЎ СТАТЫСТЫЧНАГА АЦЭНЬВАННЯ ПАРАМЕТРАЎ МАДЭЛІ АЎТАРЭГРЭСІІ ПРЫ НАЯЎНАСЦІ ЦЭНЗУРАВАННЯ СПРАВА

Ключавыя словы: МАДЭЛЬ АЎТАРЭГРЭСІІ, ЦЭНЗУРАВАННЕ, СТАТЫСТЫЧНАЕ АЦЭНЬВАННЕ ПАРАМЕТРАЎ, МЕТАД МАКСІМАЛЬНАГА ПРАЎДАПАДАБЕНСТВА, ВЫПАДКОВЫЯ ВЕКТАРА.

Аб'ект даследавання — мадэль аўтарэгрэсіі.

Мэта працы — правесці параўнальны аналіз метадаў статыстычнага ацэньвання параметраў і высветліць, якія з іх з'яўляюцца аптымальнымі.

Метады даследавання — метады тэорыі верагоднасці і матэматычнай статыстыкі, метады иммитационного і статыстычнага мадэлявання, метады матрычнага аналізу.

Вынікі працы — рэалізацыя розных метадаў статыстычнага ацэньвання параметраў мадэлі аўтарэгрэссіі, праведзены параўнальны аналіз.

Галіна выкарыстання — медыцына, эканоміка, астраномія і іншыя.

General characteristics of the work

Master's thesis, 74 pages, 19 figures, 4 tables, 16 sources, 4 appendixes.

COMPARATIVE ANALYSIS OF METHODS FOR STATISTICAL ESTIMATION OF AUTOREGRESSION MODEL PARAMETERS IN THE PRESENCE OF CENSORING ON THE RIGHT

Keywords: AUTOREGRESSION MODEL, CENSORING, STATISTICAL ESTIMATION OF PARAMETERS, MAXIMUM LIKELIHOOD METHOD, RANDOM VECTORS.

Object of study — an autoregression model.

Purpose of the work — to conduct a comparative analysis of the methods of statistical estimation of parameters and find out which of them are optimal.

Research methods — methods of probability theory and mathematical statistics, methods of simulation and statistical modeling, methods of matrix analysis.

Results of the work — implementation of various methods of statistical estimation of parameters of the autoregression model, a comparative analysis.

Scope of application — medicine, economics, astronomy and others.

Содержание

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

$\mathbb{N}$	—	множество натуральных чисел
$\mathbb{Z}$	—	множество целых чисел
$\mathbb{R}$	—	множество действительных чисел
$\mathcal{L}\{\xi\}$	—	закон распределения вероятностей ξ
$\lvert A\rvert$	—	модуль числа A, или определитель матрицы A
$[x]$	—	целая часть числа $x \in \mathbb{R}$: $[x] = \max\{a \in \mathbb{Z} : a \leq x\}$
$\lvert\lvert A\rvert\rvert$	—	норма A, если не указано обратное, то Евклидова
$\delta_{i,j}$	—	символ Кронекера: $\delta_{i,j} = \begin{cases} 1, & \text{если } i = j, \\ 0, & \text{если } i \neq j \end{cases}$
$\mathbf{I}_A(x)$	—	индикаторная функция множества A: $\mathbf{I}_A(x) = \begin{cases} 1, & \text{если } x \in A, \\ 0, & \text{если } x \notin A \end{cases}$
$\mathbf{E}\{\xi\}$	—	математическое ожидание случайной величины ξ
$\mathbf{D}\{\xi\}$	—	дисперсия случайной величины ξ
$\mathrm{Cov}\{\xi, \eta\}$	—	ковариация случайных величин ξ и η
$\mathcal{N}_N(\mu, \Sigma)$	—	N-мерный нормальный закон распределения вероятностей с вектором математического ожидания $\mu \in \mathbb{R}^N$ и ковариационной матрицей $\Sigma \in \mathbb{R}^{N \times N}$
$n_N(x\lvert\mu, \Sigma)$	—	плотность N-мерного нормального распределения $\mathcal{N}_N(\mu, \Sigma)$, вычисленная в точке $x \in \mathbb{R}^N$: $n_N(x\lvert\mu, \Sigma) = (2\pi)^{-\frac{N}{2}} \lvert\Sigma\rvert^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$
$\Phi(\cdot)$	—	функция распределения вероятностей стандартного нормального закона $\mathcal{N}_1(0,1)$: $\Phi(x) = \int_{-\infty}^{x} \varphi(t)dt$
$U_{[a,b]}$	—	непрерывный равномерный закон распределения вероятностей на отрезке $[a, b]$ с плотностью распределения вероятностей $f_X(x) = \begin{cases} \dfrac{1}{b-a}, & x \in [a,b] \\ 0, & x \notin [a,b] \end{cases}$.
E_n	—	единичная $(n \times n)$-матрица
A^T	—	транспонированная матрица A
$\det A$	—	определитель матрицы A
A^{-1}	—	матрица, обратная к квадратной матрице A
$\mathrm{AR}(p)$	—	модель авторегрессии порядка p $(p \in \mathbb{N})$
$\mathrm{grad}\, l$	—	градиент функции l
$\square$	—	символ окончания доказательства

ВВЕДЕНИЕ

Под временным рядом понимается последовательность наблюдений, полученных путем регулярных измерений определенных показателей, характеризующих состояние некоторой системы или процесса в текущий момент времени. Временные ряды возникают практически во всех сферах человеческой деятельности [4].

Основным отличием временных рядов от случайных выборок является зависимость наблюдений, во-первых, между собой, во-вторых, от номера или от времени проведения измерения. Наличие таких зависимостей зачастую значительно усложняет проведение статистического анализа, однако, позволяет решать такую важную практическую задачу, как прогнозирование [1, 2].

Как правило статистический анализ проводится в несколько этапов. На первом этапе строится так называемая априорная модель, т.е. делаются начальные предположения относительно математической модели, которая описывает наблюдаемый процесс. Затем, используя методы математической статистики, оцениваются параметры, входящие в математическую модель [10]. К сожалению, на практике полное соответствие наблюдаемых значений гипотетической априорной модели маловероятно из-за различного рода искажений в данных. Классическими видами искажений являются выбросы, пропуски, гетероскедастичность, цензурирование и другие [9]. Остановимся более подробно на таком виде искажения, как цензурирование. Цензурирование означает, что все значения временного ряда, попавшие в определенное множество значений, не наблюдаются (пропущены), тогда как наблюдения, не попавшие в это множество, известны абсолютно точно [3, 12]. Выделяют разные виды цензурирования в зависимости от множества, в котором значения пропущены: интервальное цензурирование, цензурирование слева, цензурирование справа, двустороннее цензурирование [7]. В настоящей диссертации рассматривается такой вид цензурирования, как цензурирования справа. При нем все значения временного ряда, превосходящие некоторый фиксированный уровень цензурирования, будут пропущены.

На практике цензурирование может возникать из-за наличия у измерительных приборов (датчиков) конечных пределов измерений, разладки оборудования, специфики хранения и передачи данных, нежелания респондентов давать ответы на неудобные для него вопросы и другие. Оно возникает в таких областях, как социология, метеорология, медицина, техника, финансы [3, 7, 12].

В случае случайных выборок цензурирование достаточно подробно изучено в теории надежности, однако цензурированные временные ряды по-прежнему остаются малоизученными [3]. Применяемые на практике такие простейшие

подходы, как исключение цензурированных наблюдений или замена их некоторой константой, в большинстве случаев дают смещенные и несостоятельные результаты [7]. В связи с этим возникает необходимость в разработке робастных (от англ. robust – устойчивый) к данному виду искажений методов статистического оценивания.

Одним из классических подходов к оцениванию параметров стационарного временного ряда является построение оценок максимального правдоподобия. Однако в [3] было показано, что даже в случае простейшей авторегрессии первого порядка при наличии цензурирования нахождение значения функции правдоподобия даже в одной точке – это вычислительно сложная задача. В связи с этим, в настоящей работе были рассмотрены известные в литературе [8, 12, 16] и предложены новые приемлемые с точки зрения вычислительной сложности методы построения приближённых оценок параметров цензурированного временного ряда.

В качестве одного из методов статистического оценивания параметров в рамках магистерской диссертации был рассмотрен и реализован “метод вставки” [12]. Идея заключается в том, чтобы заменить пропущенные наблюдения некоторыми значениями, после чего применить стандартные методы оценивания для данных без пропусков. Изначально предлагалось заменять все цензурированные значения некоторой константой (средним арифметическим наблюдаемых значений, уровнем цензурирования и др), однако при таком подходе результаты были неудовлетворительные [7]. Затем в 2007 году в [12] было предложено заменять цензурированные наблюдения псевдослучайными величинами из усеченного распределения, которому они должны принадлежать, согласно априорной модели. Формально, чтобы найти распределение согласно априорной модели, необходимо знать параметры этой модели. Однако это приводит к парадоксальной ситуации: чтобы оценить неизвестные параметры модели, необходимо знать распределение пропущенных наблюдений, для чего необходимо знать значения неизвестных параметров модели. Чтобы разрешить эту проблему, в работе [12] был предложен итерационный алгоритм, схожий с EM-алгоритмом, который заключается в том, что на одной итерации происходит оценивание параметров по “восстановленным” данным, а на следующей – замена цензурированных наблюдений псевдослучайными величинами на основе оценок параметров, полученных на предыдущем шаге и т.д. Данный метод является эвристическим, но, как показывает практика, может давать очень хорошие результаты. Одним из недостатков указанного метода является то, что точное нахождение закона, которому цензурированные наблюдения должны принадлежать, согласно априорной модели, может быть весьма трудоемким процессом. В связи с этим, в настоящей диссертации были предложены модификации “метода

вставки", в которых за счет игнорирования зависимостей между далеко отстоящими между собой элементами, был значительно ускорен процесс нахождения искомого закона, при этом точность оценивания параметров не пострадала.

Также был предложен и реализован приближённый метод максимального правдоподобия. Основная идея была взята из [16], где данный метод применялся для округленных данных. Идея заключается в том, что выборка разбивается на некоторое число случайных векторов s, отстающих друг от друга на заданный шаг. Эти вектора считаем независимыми, в виду того что при достаточно большом лаге, как правило, наблюдения слабо коррелируют между собой. Вариантов разбиения выборки на независимые вектора $k = \left[\frac{T}{s}\right]$, где T – размерность выборки. По каждому такому варианту разбиения строится оценка параметров модели θ_i, а окончательная оценка находится как среднее арифметическое всех найденных $\bar{\theta} = \frac{1}{k}\sum\limits_{i=1}^{k}\theta_i$.

Также был рассмотрен и реализован метод моментов для авторегрессионных моделей со случайными «пропусками» [8], который не учитывает механизм их возникновения.

В рамках настоящей диссертации был проведён сравнительный анализ всех реализованных методов. Сравнение проводилось по двум критериям: по времени построения оценок и по точности.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Математическая модель и постановка задачи

Пусть временной ряд $x_t \in \mathbb{R}$ описывается авторегрессионной моделью порядка $p(AR(p))$, согласно [11]:

$$x_t = a_1 x_{t-1} + a_2 x_{t-2} + ... + a_p x_{t-p} + u_t; \qquad -\infty < t < \infty, \tag{1.1}$$

где $a_1, ..., a_p \in \mathbb{R}, a_p \neq 0$ – коэффициенты авторегрессии такие, что все корни соответствующего характеристического многочлена лежат внутри единичного круга; u_t – последовательность независимых в совокупности одинаково распределённых случайных величин, имеющих нулевое математическое ожидание и одинаковую ограниченную дисперсию. Будем рассматривать $u_t \sim \mathcal{N}(0, \sigma^2)$, то есть имеющие нормальный закон распределения.

Согласно следствию из теоремы о стационарности временного ряда [2], представленный ряд (1.1) стационарен в широком и узком смыслах. Кроме того ряд (1.1) является гауссовским.

Наблюдения проводятся в моменты времени $t \in \{1, ..., T\}$. Будем говорить, что в момент времени t наблюдение цензурировано справа, если истинное значение временного ряда не наблюдается, а известно лишь, что $x_t > l$, где $l \in \mathbb{R}$ – заданный уровень цензурирования. Если на практике уровень цензурирования не известен, то его можно оценить максимумом среди наблюдаемых величин временного ряда.

В рамках магистерской диссертации была поставлена задача: проведение сравнительного анализа различных методов статистического оценивания параметров $(a_1, a_2, ..., a_p, \sigma^2)$ модели авторегрессии $AR(p)$ при наличии цензурирования справа. В качестве основного критерия сравнения методов использовалась вариация оценок.

$$Var(\theta) = \mathbf{E}\{(\hat{\theta} - \theta)^2\}. \tag{1.2}$$

1.2 Статистические оценки основных характеристик

Для начала приведём статистики, утверждения и оценки, которые будем использовать в дальнейшем.

Для оценивания параметров временного ряда авторегрессии без пропусков будем использовать оценки Юла-Уокера [10]. Для временного ряда $AR(p)$, система уравнений Юла-Уокера, которая связывает коэффициенты a_j, дисперсию $\mathbf{D}\{u_t\} = \sigma^2$, дисперсию $\mathbf{D}\{x_t\} = \sigma(0)$ и ковариационную функцию $\sigma(\tau) = \mathbf{E}\{(x_t - \mathbf{E}\{x_t\})(x_{t+\tau} - \mathbf{E}\{x_{t+\tau}\})\}$, имеет вид [10]:

$$\begin{cases} \sigma(0) - \sum\limits_{j=1}^{p} a_j \sigma(j) = \sigma^2, & \tau = 0, \\ \sigma(\tau) - \sum\limits_{j=1}^{p} a_j \sigma(\tau - j) = 0, & \tau \geq 1. \end{cases} \tag{1.3}$$

Для краткости записи в дальнейшем будем рассматривать лишь модель авторегрессии первого порядка ($AR(1)$). Все полученные в работе результаты могут быть обобщены на случай авторегрессии более высокого порядка.

Пусть есть некоторая реализация $X = \{x_1, ..., x_T\}, T \geq 1$ временного ряда, описываемого моделью авторегрессии первого порядка $AR(1)$.

$$x_t = ax_{t-1} + u_t, t \geq 1. \tag{1.4}$$

Для временного ряда (1.4) условие стационарности примет вид $|a| < 1$. Как и ранее полагаем, что $u_t \sim \mathcal{N}(0, \sigma^2)$.

Вычислим математическое ожидание этого временного ряда:

$$\mathbf{E}\{x_t\} = \mathbf{E}\{ax_{t-1} + u_t\} = a\mathbf{E}\{x_{t-1}\} + \mathbf{E}\{u_t\} = [\mathbf{E}\{u_t\} = 0] = a\mathbf{E}\{x_{t-1}\}, \tag{1.5}$$

а в силу того, что ряд (1.4) стационарен, его математическое ожидание не зависит от времени, т.е. $\mathbf{E}\{x_t\} = \mathbf{E}\{x_{t-1}\} = \mu$, получим: $\mu = a\mu, |a| < 1$. Такое возможно только если $\mu = \mathbf{E}\{x_t\} = 0$.

В силу того, что u_t – последовательность независимых в совокупности случайных величин и $u_t \sim \mathcal{N}(0, \sigma^2)$ получаем, что $\mathbf{E}\{u_{t-l}u_{t-k}\} = \delta_{k,l}\sigma^2$, где $\delta_{k,l} = \begin{cases} 1, k = l, \\ 0, k \neq l. \end{cases}$

Известно [10], что любой временной ряд $AR(p)$ можно представить в виде модели скользящего среднего (быть может бесконечного порядка):

$$x_t = u_t + b_1 u_{t-1} + b_2 u_{t-2} + ... + b_k u_{t-k} + ... \tag{1.6}$$

В случае авторегрессии первого порядка $AR(1)$ это представление можно записать в явном виде:

$$x_t = u_t + ax_{t-1} = u_t + a(u_{t-1} + ax_{t-2}) = u_t + au_{t-1} + a^2 x_{t-2} = u_t + au_{t-1} + a^2(u_{t-2} + \\ + ax_{t-3}) = u_t + au_{t-1} + a^2 u_{t-2} + a^3 x_{t-3} = ... = u_t + au_{t-1} + a^2 u_{t-2} + a^3 u_{t-3} + ...$$

Из представления (1.6) в частности следует, что x_t не зависит ни от какого $u_{t+\tau}, \tau > 0$. Следовательно,

$$Cov\{x_t, u_{t+\tau}\} = \mathbf{E}\{(x_t - \mathbf{E}\{x_t\})(u_{t+\tau} - \mathbf{E}\{u_{t+\tau}\})\} = \mathbf{E}\{x_t u_{t+\tau}\} = 0. \tag{1.7}$$

Вычислим дисперсию временного ряда (1.4):

$$\begin{gathered}\mathbf{D}\{x_t\} = Cov\{x_t, x_t\} = \sigma(0) = \mathbf{E}\{(u_t + au_{t-1} + a^2u_{t-2} + ...)^2\} = \mathbf{E}\{u_t^2\} + \\ +a^2\mathbf{E}\{u_{t-1}^2\} + a^4\mathbf{E}\{u_{t-2}^2\} + ... + 2a\mathbf{E}\{u_tu_{t-1}\} + 2a^2\mathbf{E}\{u_tu_{t-2}\} + ... = \\ = \sigma^2(1 + a^2 + a^4 + ...) = [|a| < 1] = \frac{\sigma^2}{1 - a^2}.\end{gathered} \tag{1.8}$$

Вычислим ковариацию временного ряда (1.4) при $\tau > 0$:

$$\begin{gathered}Cov\{x_t, x_{t+\tau}\} = Cov\{x_t, x_{t+\tau}\} = \sigma(\tau) = \mathbf{E}\{x_tx_{t+\tau}\} = \mathbf{E}\{x_t(ax_{t+\tau-1} + u_{t+\tau})\} \overset{(1.6)}{=} \\ \overset{(1.6)}{=} \mathbf{E}\{x_tax_{t+\tau-1}\} = ... = \mathbf{E}\{x_ta^\tau x_t\} = a^\tau\mathbf{E}\{x_t^2\} = a^\tau\frac{\sigma^2}{1 - a^2}.\end{gathered} \tag{1.9}$$

При $\tau < 0$ имеем $\sigma(\tau) = \sigma(-\tau)$.

Матрица автоковариации для стационарного процесса $AR(1)$, соответствующего последовательности значений $x_0, x_1, ..., x_T$ имеет вид [11]:

$$\Sigma = \frac{\sigma^2}{1 - a^2}\begin{bmatrix} 1 & a & a^2 & \cdots & a^{T-1} \\ a & 1 & a & \cdots & a^{T-2} \\ a^2 & a & 1 & \cdots & a^{T-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a^{T-1} & a^{T-2} & a^{T-3} & \cdots & 1 \end{bmatrix}. \tag{1.10}$$

Система уравнений Юла-Уокера (1.3) для временного ряда (1.4) примет вид:

$$\begin{cases}\sigma(0) - a\sigma(1) = \sigma^2, \\ \sigma(1) - a\sigma(0) = 0.\end{cases} \tag{1.11}$$

Теперь воспользуемся тем, что математическое ожидание данного временного ряда равно нулю, как показано выше, и заменим в (1.3) ковариационную функцию $\sigma(\tau)$ на её статистическую оценку [7]:

$$c_\tau = c_{-\tau} = \frac{1}{T - \tau}\sum_{t=1}^{T-\tau} x_tx_{t+\tau}. \tag{1.12}$$

В общем случае для оценки параметров $(a_1, a_2, ..., a_p)$ достаточно p уравнений Юла-Уокера, где $\tau \neq 0$:

$$c_\tau - \sum_{j=1}^{p} a_jc_{\tau-j} = 0, \qquad \tau = 1, ..., p. \tag{1.13}$$

В случае $p = 1$ имеем одно уравнение: $c_1 - \widehat{a}c_0 = 0$, откуда $\widehat{a} = \frac{c_1}{c_0}$.

Используя полученную оценку, оценим дисперсию σ^2 из первого уравнения (1.11) или (1.3) при $\tau = 0$:

$$\widehat{\sigma^2} = c_0 - \widehat{a}c_1. \tag{1.14}$$

В случае, когда все значения временного ряда известны, полученные нами оценки являются состоятельными [11].

1.3 Метод вставки

Теперь опишем "метод вставки" подробно согласно [12]. Предположим у нас есть некоторая реализация $X = \{x_1, ..., x_t\}$, $T \geq 1$, временного ряда, описываемая (1.4). В этой выборке часть наблюдений цензурировано, обозначим их как X_c, и нам известен уровень цензурирования, который обозначим как l. Наблюдения, которые присутствуют в выборке (наблюдаемые), обозначим через X_o.

Данный метод итерационный и на k-ой итерации выполняются следующие шаги:

1. Пусть $a^{(k)}$, $(\sigma^2)^{(k)}$ – текущие оценки параметров модели AR(1).

2. Вектор X_c заменяем вектором, сгенерированным из псевдослучайных величин из усеченного нормального закона распределения при условии фиксированных значений X_o.

$$X_c^{(k)} \sim p(X_c|X_o; a^{(k)}, (\sigma^2)^{(k)}), \tag{1.15}$$

   где $p(x)$– условная плотность усечённого нормального закона распределения вероятностей, интервал возможных значений которого $[l, +\infty)$.

3. После "восстановления" всех цензурированных значений на шаге 2 мы получаем классическую ситуацию, когда все значения временного ряда присутствуют (возможно они отличаются от истинных значений), поэтому для оценки параметров можно воспользоваться классическими оценками, например, оценками Юла-Уокера. Оцениваем параметры модели по методу Юла-Уокера и получаем оценки $a^{(k+1)}$, $(\sigma^2)^{(k+1)}$.

4. Повторяем шаги 1-3 до тех пор пока $\|a^{(k+1)} - a^{(k)}\| + \|(\sigma^2)^{(k+1)} - (\sigma^2)^{(k)}\| > \varepsilon$, где ε – некоторая наперёд заданная величина.

Изначально в методах вставки предполагалось заменять пропущенные (или цензурированные) наблюдения некоторыми статистиками от присутству-

ющих наблюдений (например, средним арифметическим присутствующих наблюдений), однако даже в простейших случаях такая замена сильно смещала оценки дисперсии и ковариационной функции [7]. В предложенном методе цензурированные случайные величины заменяются на псевдослучайные значения из необходимого распределения. Тем самым делается попытка уменьшить смещение дисперсии и ковариационной функции. Отметим, что данный метод эвристический, не имеет строгого математического обоснования сходимости или конечности. Однако на практике во многих случаях он даёт неплохие результаты [12].

Начальные значения параметров $a^{(1)}$ и $(\sigma^2)^{(1)}$ оценивались следующим образом:

1. Все цензурированные значения заменяем на уровень цензурирования l.

2. Получаем классическую ситуацию, когда все значения временного ряда известны, поэтому применимы оценки Юла-Уокера. Оцениваем полученный временной ряд, предложенными оценками.

В нашем случае, так как временной ряд является гауссовским, то $p(X) = n_T(x_1, ..., x_T|M, \Sigma)$ – плотность T-мерного нормального закона распределения с математическим ожиданием $M = (0, .., 0)^T \in \mathbb{R}^T$ и ковариационной $(T \times T)$ матрицей Σ (1.10). Кроме того, из свойств многомерного нормального закона распределения известно, что если $x \sim \mathcal{N}_T(\mu, \sigma^2)$, то применима теорема об условном распределении вероятностей гауссовских случайных векторов.

Теорема 1.1. [10] *Пусть случайный N-вектор $x = \begin{pmatrix} x^{(1)} \\ x^{(2)} \end{pmatrix} \in \mathbb{R}^N$ имеет невырожденное нормальное распределение $\mathcal{N}_N(\mu, \Sigma)$, где $\mu = \begin{pmatrix} \mu^{(1)} \\ \mu^{(2)} \end{pmatrix}$, $\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$, и разбит на два подвектора $x^{(1)} \in \mathbb{R}^m$, $x^{(2)} \in \mathbb{R}^{N-m}$. Тогда условное распределение случайного вектора $x^{(1)}$ при фиксированном значении вектора $x^{(2)}$ – m-мерное нормальное со следующими параметрами, которые задаются формулами:*

$$\mathcal{L}\{x^{(1)}|x^{(2)}\} = \mathcal{N}_m(\mu_{1|2}, \Sigma_{11|2}), \tag{1.16}$$

$$\mu_{1|2} = \mu^{(1)} + \Sigma_{12}\Sigma_{22}^{-1}(x^{(2)} - \mu^{(2)}), \tag{1.17}$$

$$\Sigma_{11|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}. \tag{1.18}$$

В нашем случае вектор $x^{(1)}$ – это вектор цензурированных значений выборки (X_C), вектор $x^{(2)}$ – вектор, состоящий из наблюдаемых значений выборки

(X_O).

Как видно из условия теоремы, разбиение вектора на подвектора идёт последовательно, однако в нашем случае компоненты подвекторов идут не по порядку. Поэтому, чтобы наши данные подходили под условия теоремы, можно использовать матрицу перестановок P так, чтобы разделить вектор данных на наблюдаемые и цензурированные значения. Тогда PX также удовлетворяет многомерному нормальному распределению с параметрами:

$$P = \begin{pmatrix} P_C \\ P_O \end{pmatrix}; PX \sim \mathcal{N}_N \left(\mu P E_N, \begin{pmatrix} P_C \Sigma P_C^T & P_C \Sigma P_O^T \\ P_O \Sigma P_C^T & P_O \Sigma P_O^T \end{pmatrix} = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \right).$$

Заметим, что в формуле (1.17) нужно вычислить обратную матрицу Σ_{22}^{-1}. Рассмотрим эту задачу более подробно.

Для удобства введем следующие обозначения $a_k = a^{t_{k+1}-t_k}$, $t_1 < t_2 < \\ < \ldots < t_n$, $n = N - m$, тогда матрица Σ_{22} будет иметь вид:

$$\Sigma_{22} = \frac{\sigma^2}{1-a^2} \begin{bmatrix} 1 & a_1 & a_1 a_2 & \cdots & a_1 a_2 \ldots a_{n-1} \\ a_1 & 1 & a_2 & \cdots & a_2 \ldots a_{n-1} \\ a_1 a_2 & a_2 & 1 & \cdots & a_3 \ldots a_{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 a_2 \ldots a_{n-1} & a_2 \ldots a_{n-1} & a_3 \ldots a_{n-1} & \cdots & 1 \end{bmatrix}. \tag{1.19}$$

Теорема 1.2. *Обратная матрица к матрице из (1.19) имеет вид:*

$$\Sigma_{22}^{-1} = \frac{1-a^2}{\sigma^2} \begin{bmatrix} \frac{1}{1-a_1^2} & -\frac{a_1}{1-a_1^2} & 0 & \cdots & 0 \\ -\frac{a_1}{1-a_1^2} & \frac{1-a_1^2 a_2^2}{(1-a_1^2)(1-a_2^2)} & -\frac{a_2^2}{1-a_2^2} & \cdots & 0 \\ 0 & -\frac{a_2^2}{1-a_2^2} & \frac{1-a_2^2 a_3^2}{(1-a_2^2)(1-a_3^2)} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{1-a_{n-1}^2} \end{bmatrix}. \tag{1.20}$$

Доказательство. Доказательство будем проводить по методу математической индукции.

База индукции. Проверим для случая $n = 2$. По определению обратной

матрицы $\Sigma_{22}^{-1}\Sigma_{22} = E_n$. В нашем случае:

$$\Sigma_{22}^{-1}\Sigma_{22} = \frac{1-a^2}{\sigma^2}\begin{bmatrix} \frac{1}{1-a_1^2} & -\frac{a_1}{1-a_1^2} \\ -\frac{a_1}{1-a_1^2} & \frac{1}{1-a_1^2} \end{bmatrix}\frac{\sigma^2}{1-a^2}\begin{bmatrix} 1 & a_1 \\ a_1 & 1 \end{bmatrix} = \\ = \frac{1-a^2}{\sigma^2}\frac{\sigma^2}{1-a^2}\begin{bmatrix} \frac{1}{1-a_1^2} - \frac{a_1^2}{1-a_1^2} & \frac{a_1}{1-a_1^2} - \frac{a_1}{1-a_1^2} \\ \frac{a_1}{1-a_1^2} - \frac{a_1}{1-a_1^2} & -\frac{a_1^2}{1-a_1^2} + \frac{1}{1-a_1^2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{1.21}$$

Допустим, что утверждение верно для $n = r-1$. Докажем для случая $n = r$. Докажем также по определению. Представим матрицы Σ_{22}^{-1} и Σ_{22} в следующем виде:

$$\Sigma_{22}^{-1} = \frac{1-a^2}{\sigma^2}\begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix}, \Sigma_{22} = \frac{\sigma^2}{1-a^2}\begin{bmatrix} A & B \\ C & D \end{bmatrix}, \tag{1.22}$$

$$A_1 = \frac{1}{1-a_1^2} \in \mathbb{R};\ B_1 = C_1^T = \left(-\frac{a_1}{1-a_1^2},\ 0,\ 0,\ ...,\ 0\right) \in \mathbb{R}^{r-1};$$

$$D_1 = \begin{bmatrix} \frac{1-a_1^2a_2^2}{(1-a_1^2)(1-a_2^2)} & -\frac{a_2^2}{1-a_2^2} & 0 & \cdots & 0 \\ -\frac{a_2^2}{1-a_2^2} & \frac{1-a_2^2a_3^2}{(1-a_2^2)(1-a_3^2)} & -\frac{a_3^2}{1-a_3^2} & \cdots & 0 \\ -\frac{a_3^2}{1-a_3^2} & \frac{1-a_3^2a_4^2}{(1-a_3^2)(1-a_4^2)} & -\frac{a_4^2}{1-a_4^2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{1-a_{r-1}^2} \end{bmatrix} \in \mathbb{R}^{(r-1)(r-1)}; \tag{1.23}$$

$$A = 1 \in \mathbb{R};\quad B = C^T = (a_1,\ a_1a_2,\ a_1a_2a_3,\ ...,\ a_1..a_{r-1}) \in \mathbb{R}^{r-1};$$

$$D = \begin{bmatrix} 1 & a_2 & a_2a_3 & \cdots & a_2...a_{r-1} \\ a_2 & 1 & a_3 & \cdots & a_3...a_{r-1} \\ a_2a_3 & a_3 & 1 & \cdots & a_4...a_{r-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_2...a_{r-1} & a_3...a_{r-1} & a_4...a_{r-1} & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{(r-1)(r-1)}. \tag{1.24}$$

Тогда $\Sigma_{22}^{-1}\Sigma_{22} = \frac{1-a^2}{\sigma^2}\begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix}\frac{\sigma^2}{1-a^2}\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A_1A + B_1C & A_1B + B_1D \\ C_1A + D_1C & C_1B + D_1D \end{bmatrix}$, при чем $A_1B + B_1D = (C_1A + D_1C)^T$.

Нужно показать, что $A_1A + B_1C = 1, A_1B + B_1D = \overrightarrow{0}, C_1B + D_1D = E_{r-1}$.

1. $A_1A + B_1C = \frac{1}{1-a_1^2} + \frac{-a_1}{1-a_1^2}a_1 = 1$;

2. $A_1B + B_1D = \left(\frac{a_1}{1-a_1^2} - \frac{a_1}{1-a_1^2},\ \frac{a_1a_2}{1-a_1^2} - \frac{a_1}{1-a_1^2}a_2,\ ...,\ \frac{1}{1-a_1^2}a_1...a_{r-1} - \frac{a_1}{1-a_1^2}a_2...a_{r-1}\right) =$

$= (0,\ 0,\ ...,\ 0)$;

3. Проведем дополнительные вычисления. В силу индукционного предположения обратная матрица D имеет вид (1.20). Тогда матрица $D_1 = D^{-1} + D^*$, где

$$D^* = \begin{bmatrix} \frac{1-a_1^2a_2^2}{(1-a_1^2)(1-a_2^2)} - \frac{1}{1-a_2^2} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

Тогда $C_1B + D_1D = C_1B + (D^{-1} + D^*)D = C_1B + E_{r-1} + D^*D$, т.е. нужно показать, что $C_1B + D^*D$ – нулевая матрица.

$$C_1B + D^*D = \begin{bmatrix} -\frac{a_1^2}{1-a_1^2} & -\frac{a_1^2a_2}{1-a_1^2} & \cdots & -\frac{a_1^2a_2...a_{r-1}}{1-a_1^2} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} +$$

$$+ \begin{bmatrix} \frac{1-a_1^2a_2^2}{(1-a_1^2)(1-a_2^2)} - \frac{1}{1-a_2^2} & \cdots & a_2...a_{r-1}\left(\frac{1-a_1^2a_2^2}{(1-a_1^2)(1-a_2^2)} - \frac{1}{1-a_2^2}\right) \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{1-a_1^2a_2^2-1+a_1^2-a_1^2(1-a_2^2)}{(1-a_1^2)(1-a_2^2)} & \cdots & \frac{-a_1^2a_2...a_{r-1}(1-a_2^2)+a_2...a_{r-1}(1-a_1^2a_2^2-1+a_1^2)}{(1-a_1^2)(1-a_2^2)} \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} =$$

$$= \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

□

Для удобства введём обозначение: $\mathcal{N}(\mu, \sigma^2; [l; +\infty))$ – нормальный закон распределения с математическим ожиданием μ и дисперсией σ^2, усеченный на интервале $[l; +\infty)$.

На 2 шаге “метода вставки” нужно сгенерировать $m-$вектор X из невырожденного нормального распределения $X \sim \mathcal{N}_m(\mu_{1|2}, \Sigma_{11|2})$, каждая компонен-

та которого должна быть больше уровня цензурирования. Чтобы достичь этих 2 условий будем действовать следующим образом:

1. Так как матрица $\Sigma_{11|2}$ - симметричная и положительно определенная, то для нее справедливо разложение Холецкого [13] $\Sigma_{11|2} = CC^T$, где C – нижняя треугольная матрица со строго положительными элементами на диагонали.

2. Генерируем m независимых одномерных случайных величин y_j, $j \in \{1, ..., m\}$, при чем каждое $y_j \sim \mathcal{N}(0, 1; [b_j; +\infty))$, где b_j – j-ая компонента вектора $C^{-1}(L - \mu_{1|2})$, $L = \begin{pmatrix} l \\ \vdots \\ l \end{pmatrix} \sim \mathbb{R}^m$. Вопрос генерации псевдослучайной величины из усеченного нормального распределения бедут рассмотрен позже.

3. Вычисляем вектор $X = CY + \mu_{1|2}$. Так как $Y \sim \mathcal{N}_m(0, E_m)$, то $X \sim \mathcal{N}_m(\mu_{1|2}, \Sigma_{11|2})$. Покажем это:

$$\mathbf{E}\{X\} = \mathbf{E}\{CY + \mu_{1|2}\} = C\mathbf{E}\{Y\} + \mu_{1|2} = [\mathbf{E}\{Y\} = 0] = \mu_{1|2}.$$

$$Cov(CY, CY) = \mathbf{E}\{(CY)(CY)^T\} = C\mathbf{E}\{YY^T\}C^T = CE_mC^T = CC^T = \Sigma_{11|2}.$$

Также для m-ой компоненты вектора X, имеем $x_j = (CY + \mu_{1|2})_j > b_j = (CC^{-1}(L - \mu_{1|2}) + \mu_{1|2})_j = (E_m(L - \mu_{1|2}) + \mu_{1|2})_j = l_j = l$, т.е. $x_j > l, l \in L$.

Рассмотренный “метод вставки” был реализован и результаты его работы приведены в Разделе 2. В целом, метод показывает неплохие результаты, даже в случаях, когда цензурировано 25% наблюдений. Основной недостаток рассмотренного метода заключается в том, что на каждой итерации необходимо совершать операции над матрицами больших размерностей (1.18): перемножать, обращать, а они являются весьма трудоемкими задачами.

В дальнейшем общий случай “метода вставки” для краткости записи будем называть “all observations”.

Для ускорения работы метода, было предложено несколько модификаций, основанных на том, что с ростом лага в модели авторегрессии зависимость между наблюдениями убывает очень быстро и начиная с некоторого момента ей можно пренебречь.

1.3.1 Восстановление по одному наблюдению из предыстории

Найдём закон распределения цензурированного наблюдения x_{t+i} $(i \geq 1)$ в зависимости от x_t. Для этого рассмотрим вектор $x = (x_{t+i}, x_t)$. Предположим, что значение x_t наблюдается, а последующие за ним i наблюдений $x_{t+1}, ..., x_{t+i}$

цензурированы. Условные математическое ожидание и матрица автоковариации рассчитываются по формулам:

$$\mathcal{L}\{x\} = \mathcal{N}_2(\mu^{(1)}, \Sigma^{(1)});\ \ \mu^{(1)} = \begin{pmatrix}0\\0\end{pmatrix};\ \ \Sigma^{(1)} = \frac{\sigma^2}{1-a^2}\begin{pmatrix}1 & a^i\\ a^i & 1\end{pmatrix}; \tag{1.25}$$

$$\mathcal{L}\{x_{t+i}|x_t\} = \mathcal{N}(\mu^{(1)}_{1|2}, \Sigma^{(1)}_{11|2}),\quad i \geqslant 1; \tag{1.26}$$

$$\mu^{(1)}_{1|2} = a^i x_t; \tag{1.27}$$

$$\Sigma^{(1)}_{11|2} = \frac{\sigma^2}{1-a^2}(1-a^{2i}). \tag{1.28}$$

Заменим шаг 2 описанного ранее метода вставки следующим образом:

2.1) Выберем $x_{t+i} \in X_C$, где x_t – это ближайшая во временном ряду наблюдаемая предыстория ($x_t \in X_O$).

2.2) Заменяем цензурированное значение x_{t+i} псевдослучайной величиной из $\mathcal{N}(\mu^{(1)}_{1|2}, \Sigma^{(1)}_{11|2}; [l; +\infty))$.

2.3) Повторить шаги 2.1 и 2.2 для всех $x_{t+i} \in X_C$.

Т.е. фактически вместо замены сразу всего вектора X_C и вычисления параметров его распределения, мы заменяем по одному значению из X_C, при этом вычисления параметров распределения очень простые.

Визуализация данной модификации представлена на (Рисунок 1.1).

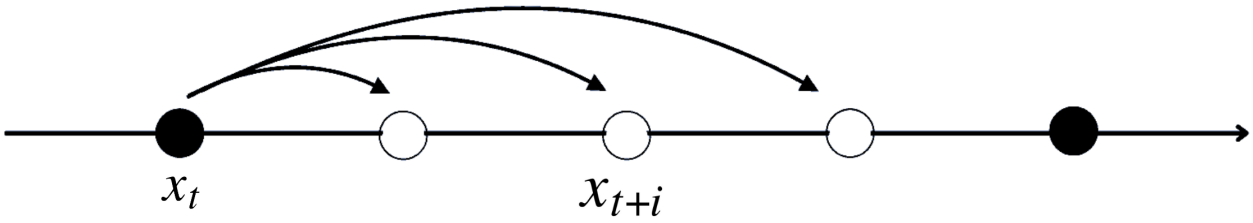

Рисунок 1.1: Визуализация модификации “1 before”

Предложенная модификация метода вставки была реализована, однако она показала не очень хорошие результаты (см. гл. 2 раздел 2.2, название метода - “1 before”).

Предположительно, основная причина этого в том, что если у нас есть несколько подряд идущих цензурированных наблюдений $x_{t+1}, .., x_{t+i-1}$, то мы “восстанавливаем” каждое из них по наблюдаемой предыстории x_t, при этом никак не учитывая зависимость между этими цензурированными значениями, которая должна быть согласно модельным предположениям. Чтобы исправить этот недостаток рассмотрим ещё одну модификацию метода вставки, в котором заменим шаг 2 на следующий:

2.1) $t = 1$.

2.2) Увеличиваем t до тех пор, пока x_{t+1} не станет цензурировано ($x_{t+1} \in X_C$) или $t = N$.

2.3) Заменяем x_{t+1} псевдослучайной величиной из $\mathcal{N}(\mu^{(1)}_{1|2}, \Sigma^{(1)}_{11|2}; [l; +\infty))$ при $i = 1$, при этом, если $x_t \in X_C$, то используем значение, которое было для него уже сгенерировано.

2.4) Если $t = N$ конец, иначе переходим к шагу 2.2.

Найдем значения параметров (1.25) – (1.28) в явном виде при $i = 1$:

$$\mathcal{L}\{x\} = \mathcal{N}_2(\mu^{(1)}, \Sigma^{(1)});\ \ \mu^{(1)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix};\ \ \Sigma^{(1)} = \frac{\sigma^2}{1-a^2}\begin{pmatrix} 1 & a \\ a & 1 \end{pmatrix}; \tag{1.29}$$

$$\mathcal{L}\{x_{t+1}|x_t\} = \mathcal{N}(\mu^{(1)}_{1|2}, \Sigma^{(1)}_{11|2}); \tag{1.30}$$

$$\mu^{(1)}_{1|2} = ax_t; \tag{1.31}$$

$$\Sigma^{(1)}_{11|2} = \frac{\sigma^2}{1-a^2}(1-a^2) = \sigma^2. \tag{1.32}$$

Визуализация данной модификации представлена на (Рисунок 1.2).

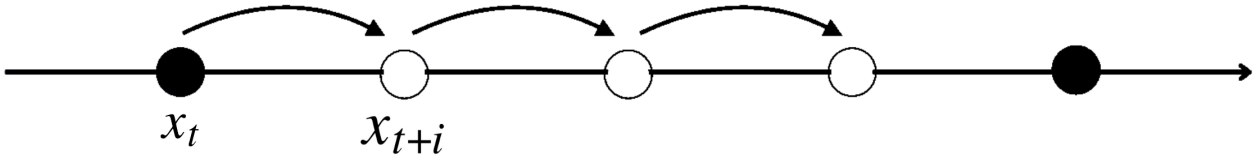

Рисунок 1.2: Визуализация модификации “1 any before”

Данная предложенная модификация показала значительно более хорошие результаты по сравнению с предыдущей модификацией (см. гл. 2 раздел 2.2, название метода - “1 any before”).

1.3.2 Восстановление по двум наблюдениям

Найдём закон распределения цензурированного наблюдения x_{t+i} ($i \geq 1$) в зависимости от x_t и x_{t+j}. Рассмотрим несколько возможных случаев расположения наблюдений x_t и x_{t+j} от x_{t+i}.

1. Рассмотрим вектор $x = (x_{t+i}, x_t, x_{t+j})$, $j > i$. Предположим, что значение x_t наблюдается, а последующие за ним $j-1$ наблюдения $x_{t+1}, \ldots, x_{t+j-1}$ цензурированы. Будем заменять i-ое наблюдение $i < j$, используя таким образом

одно предыдущее и одно последующее значения.

$$\mathcal{L}\{x\} = \mathcal{N}_3(\mu^{(2)}, \Sigma^{(2)});\ \mu^{(2)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};\ \Sigma^{(2)} = \frac{\sigma^2}{1-a^2}\begin{pmatrix} 1 & a^i & a^{i-j} \\ a^i & 1 & a^j \\ a^{j-i} & a^j & 1 \end{pmatrix}; \quad (1.33)$$

$$\mathcal{L}\{x_{t+i}|x_t, x_{t+j}\} = \mathcal{N}(\mu^{(2)}_{1|2}, \Sigma^{(2)}_{11|2}),\ \ i \geqslant 1, j \geqslant 2; \quad (1.34)$$

$$\mu^{(2)}_{1|2} = \frac{1}{1-a^{2j}}((a^i - a^{2j-i})x_t + (a^{j-i} - a^{j+i})x_{t+j}); \quad (1.35)$$

$$\Sigma^{(2)}_{11|2} = \frac{\sigma^2}{(1-a^2)(1-a^{2j})}(1 - a^{2i} - a^{2j-2i} + a^{2j}). \quad (1.36)$$

Заменим шаг 2 описанного ранее метода вставки следующим образом:

2.1) Выберем $x_{t+i} \in X_C$, где x_t – это ближайшая во временном ряду наблюдаемая предыстория ($x_t \in X_O$), x_{t+j} – ближайшее значение в будущем ($x_{t+j} \in X_O$).

2.2) Заменяем цензурированное значение x_{t+i} псевдослучайной величиной из $\mathcal{N}(\mu^{(2)}_{1|2}, \Sigma^{(2)}_{11|2}; [l; +\infty))$.

2.3) Повторить шаги 2.1 и 2.2 для всех $x_{t+i} \in X_C$.

Визуализация данной модификации представлена на (Рисунок 1.3).

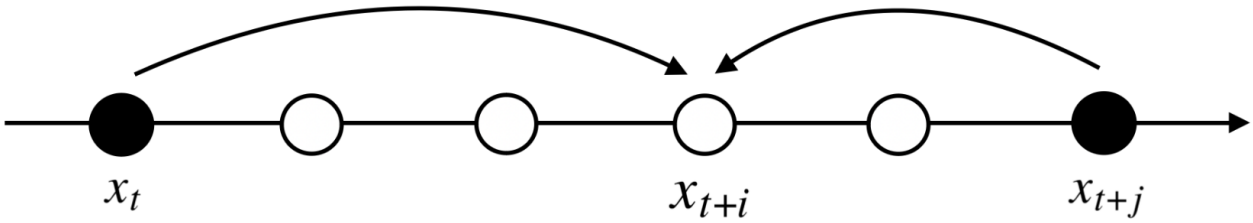

Рисунок 1.3: Визуализация модификации “1 before and 1 after”

Предложенная модификация метода вставки была реализована, однако она показала не очень хорошие результаты (см. гл. 2 раздел 2.2, название метода - “1 before and 1 after”).

Чтобы исправить этот недостаток рассмотрим ещё одну модификацию метода вставки, в котором заменим шаг 2 на следующий:

2.1) $t = 1$.

2.2) Увеличиваем t до тех пор, пока x_{t+1} не станет цензурировано ($x_{t+1} \in X_C$) или $t = N$.

2.3) Заменяем x_{t+1} псевдослучайной величиной из $\mathcal{N}(\mu^{(2)}_{1|2}, \Sigma^{(2)}_{11|2}; [l; +\infty))$ при $i = 1$, при этом, если $x_t \in X_C$, то используем значение, которое было

для него уже сгенерировано. Выбор x_{t+j} остаётся прежним.

2.4) Если $t = N$ конец, иначе переходим к шагу 2.2.

Найдем значения параметров (1.33) – (1.36) в явном виде при $i = 1$:

$$\mathcal{L}\{x\} = \mathcal{N}_3(\mu^{(2)}, \Sigma^{(2)});\ \mu^{(2)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};\ \Sigma^{(2)} = \frac{\sigma^2}{1-a^2}\begin{pmatrix} 1 & a & a^{1-j} \\ a & 1 & a^j \\ a^{j-1} & a^j & 1 \end{pmatrix}; \quad (1.37)$$

$$\mathcal{L}\{x_{t+1}|x_t, x_{t+j}\} = \mathcal{N}(\mu^{(2)}_{1|2}, \Sigma^{(2)}_{11|2}), \quad j \geqslant 2; \quad (1.38)$$

$$\mu^{(2)}_{1|2} = \frac{1}{1-a^{2j}}((a - a^{2j-1})x_t + (a^{j-1} - a^{j+1})x_{t+j}); \quad (1.39)$$

$$\Sigma^{(2)}_{11|2} = \frac{\sigma^2}{(1-a^2)(1-a^{2j})}(1 - a^2 - a^{2j-2} + a^{2j}). \quad (1.40)$$

Визуализация данной модификации представлена на (Рисунок 1.4).

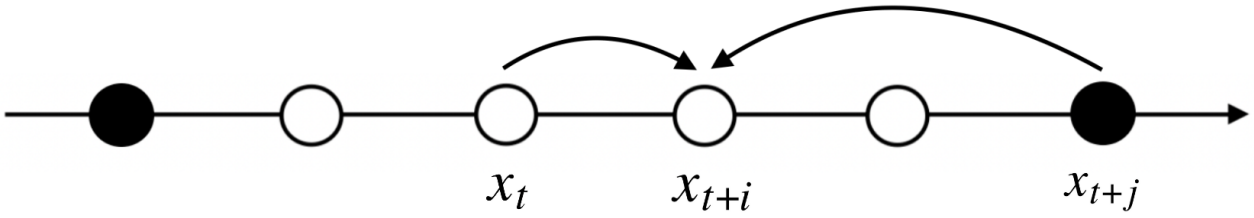

Рисунок 1.4: Визуализация модификации “1 any before and 1 after”

Предложенная модификация метода вставки была реализована, однако она показала не очень хорошие результаты (см. гл. 2 раздел 2.2, название метода - “1 any before and 1 after”).

2. В ходе работы были рассмотрены и реализованы другие варианты расположения наблюдений: восстановление проводилось по 2 наблюдениям из предыстории (см. гл. 2 раздел 2.2, название метода - “2 before”), по 2 любым наблюдениям из предыстории (см. гл. 2 раздел 2.2, название метода - “2 any before”). Рассмотренный вектор $x = (x_{t+i}, x_t, x_{t+j})$, $j < i$, условные математическое ожидание и матрица автоковариации рассчитывались по следующим формулам (для случая любых наблюдений нужно подставить $j = 1, i = 2$) :

$$\mathcal{L}\{x\} = \mathcal{N}_3(\mu^{(3)}, \Sigma^{(3)});\ \mu^{(3)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix};\ \Sigma^{(3)} = \frac{\sigma^2}{1-a^2}\begin{pmatrix} 1 & a^i & a^{i-j} \\ a^i & 1 & a^j \\ a^{i-j} & a^j & 1 \end{pmatrix}; \quad (1.41)$$

$$\mathcal{L}\{x_{t+i}|x_t, x_{t+j}\} = \mathcal{N}(\mu^{(3)}_{1|2}, \Sigma^{(3)}_{11|2}), \quad j \geqslant 1, i \geqslant 2; \tag{1.42}$$

$$\mu^{(3)}_{1|2} = \frac{1}{1-a^{2j}}(a^{i-j} - a^{i+j})x_{t+j}; \tag{1.43}$$

$$\Sigma^{(3)}_{11|2} = \frac{\sigma^2}{(1-a^2)(1-a^{2j})}(1 - a^{2j} - a^{2i-2j} + a^{2i}). \tag{1.44}$$

Визуализация этих модификаций представлены на (Рисунок 1.5 – Рисунок 1.6).

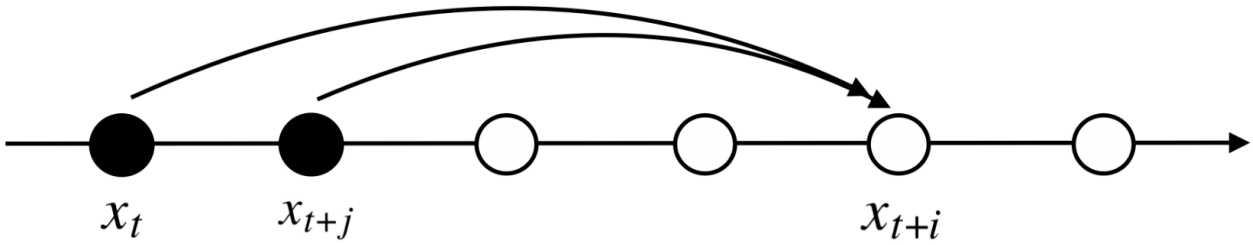

Рисунок 1.5: Визуализация модификации “2 before”

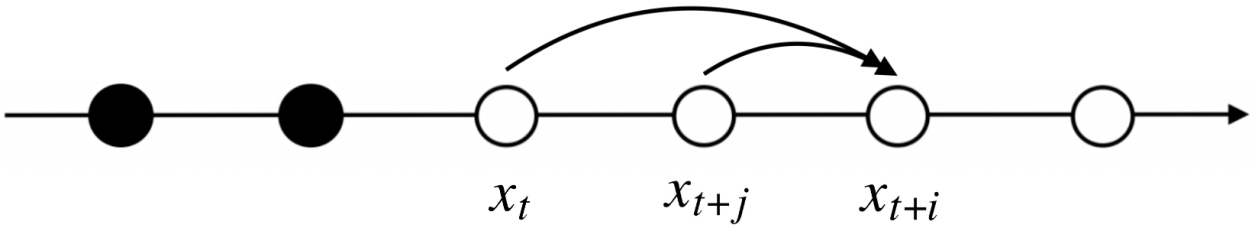

Рисунок 1.6: Визуализация модификации “2 any before”

3. В последней модификации восстановление проводилось по 2 наблюдениям в будущем (см. гл. 2 раздел 2.2, название метода - “2 after”). Рассмотренный вектор $x = (x_t, x_{t+i}, x_{t+j})$, $j > i$, условные математическое ожидание и матрица автоковариации рассчитывались по следующим формулам :

$$\mathcal{L}\{x\} = \mathcal{N}_3(\mu^{(4)}, \Sigma^{(4)}); \;\; \mu^{(4)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}; \;\; \Sigma^{(4)} = \frac{\sigma^2}{1-a^2}\begin{pmatrix} 1 & a^i & a^j \\ a^i & 1 & a^{j-i} \\ a^j & a^i & 1 \end{pmatrix}; \tag{1.45}$$

$$\mathcal{L}\{x_t|x_{t+i}, x_{t+j}\} = \mathcal{N}(\mu^{(4)}_{1|2}, \Sigma^{(4)}_{11|2}), \quad i \geqslant 1, j \geqslant 2; \tag{1.46}$$

$$\mu^{(4)}_{1|2} = \frac{1}{1-a^{2(j-i)}}(a^i - a^{2j-i})x_{t+i}; \tag{1.47}$$

$$\Sigma^{(4)}_{11|2} = \frac{\sigma^2}{(1-a^2)(1-a^{2(j-i)})}(1 - a^{2i} - a^{2j-2i} + a^{2j}). \tag{1.48}$$

Визуализация данной модификации представлена на (Рисунок 1.7).

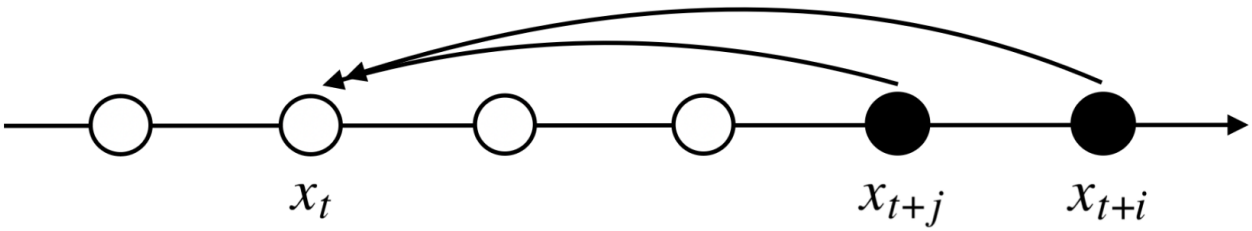

Рисунок 1.7: Визуализация модификации “2 after”

1.3.3 Восстановление по трём наблюдениям

Также в ходе дипломной работы был рассмотрено и реализовано восстановление по 3 наблюдениям: по 2 наблюдениям из предыстории и 1 наблюдению из будущего (см. гл. 2 раздел 2.2, название метода - “2 before and 1 after”). Рассмотренный вектор $x = (x_{t+m}, x_t, x_{t+h}, x_{t+q}), q > m > h$, условные математическое ожидание и матрица автоковариации рассчитывались по следующим формулам:

$$\begin{gathered}\mathcal{L}\{x\} = \mathcal{N}_4(\mu^{(5)}, \Sigma^{(5)});\ \ \mu^{(5)} = \begin{pmatrix}0\\0\\0\\0\end{pmatrix};\\ \Sigma^{(5)} = \frac{\sigma^2}{1-a^2}\begin{pmatrix}1 & a^m & a^{m-h} & a^{q-m}\\ a^m & 1 & a^h & a^q\\ a^{m-h} & a^h & 1 & a^{q-h}\\ a^{q-m} & a^q & a^{q-h} & 1\end{pmatrix};\end{gathered} \tag{1.49}$$

$$\mathcal{L}\{x_{t+m}|x_t, x_{t+h}, x_{t+q}\} = \mathcal{N}(\mu^{(5)}_{1|2}, \Sigma^{(5)}_{11|2}), \quad h \geqslant 1, m \geqslant 2, q \geqslant 3; \tag{1.50}$$

$$\begin{gathered}\mu^{(5)}_{1|2} = \frac{1}{1-a^{2q-2h}-a^{2h}+a^{2q}}((a^{m-h}-a^{m+h}+a^{2q+h-m}-a^{2q-h-m})x_{t+h}+\\ +(a^{q+m}-a^{q-m}+a^{q+m-2h}-a^{q-m+2h})x_{t+q});\end{gathered} \tag{1.51}$$

$$\begin{gathered}\Sigma^{(5)}_{11|2} = \frac{\sigma^2}{(1-a^2)(1-a^{2q-2h}-a^{2h}+a^{2q})}(1+3a^{2q}-3a^{2q-2h}-a^{2h}+\\ +a^{2q-2m}-a^{2q+2h-2m}-a^{2m}+a^{2m-2h}).\end{gathered} \tag{1.52}$$

Визуализация данной модификации представлена на (Рисунок 1.8).

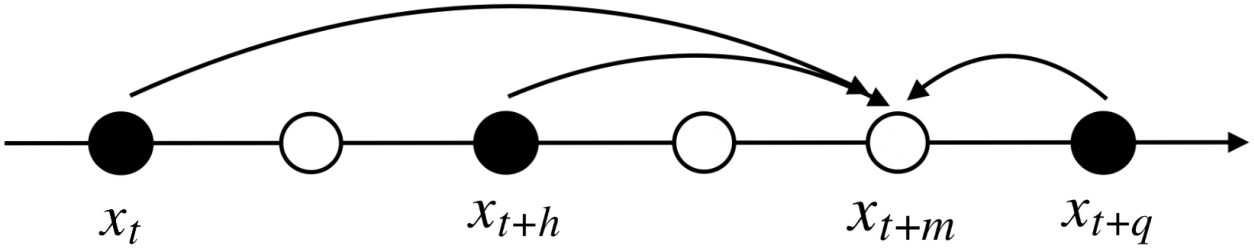

Рисунок 1.8: Визуализация модификации “2 before and 1 after”

Таким образом, мы рассмотрели частные случаи вектора $x^{(2)}$, когда он состоит из 1, 2 и 3 наблюдений.

Замечание 1.1. Стоит отметить, что во всех предложенных модификациях возможны ситуации, когда замены из шага 2.2 невозможны ввиду отсутствия необходимых значений из предыстории и/или из будущего, например, $x_1, x_T \in X_C$ для случая востановления по двум наблюдения. В этом случае заменяем x_1, x_T псевдослучайными величинами из безусловного нормального закона распределения $\mathcal{N}(a^{(k)}, (\sigma^2)^{(k)}; [l; +\infty))$.

1.4 Алгоритмы генерации псевдослучайных величин из усеченного нормального закона распределения

В шаге 2 алгоритма метода вставки нам необходимо сгенерировать случайную величину из $\mathcal{N}_1(\mu, \Sigma; [l; +\infty))$. Нами было рассмотрено 2 способа генерации.

В первом случае алгоритм генерации следующий:

1. Генерируем $\xi \sim \mathcal{N}(\mu, \Sigma)$.

2. Если $\xi > l$, вернуть ξ, иначе вернуться на шаг 1).

Данный алгоритм является весьма простым в реализации. Однако, при большом значении l, вероятность, что $P\{\xi > l\} = 1 - \Phi(l)$ становится очень маленькой, поэтому среднее число итераций $\frac{1}{1-\Phi(l)}$ становится слишком большим. Поэтому в случае большого l лучше использовать следующий алгоритм, представленный в [14]. Его суть состоит в следующем:

1. Генерируем случайную велицину $z \sim Exp(\alpha^*, l)$, где $Exp(\alpha^*, l)$ - преобразованное экспоненциальное распределение с плотностью распределения вероятностей $g(z|\alpha, l) = \alpha e^{-\alpha(z-l)} I_{z \geq l}$; α^* вычисляется по формуле $\alpha^*(l) = \frac{l+\sqrt{l^2+4}}{2}$, l - уровень цензурирования;

2. Вычисляем $\varrho(z) = e^{\frac{-(z-\alpha^*)^2}{2}}$;

3. Генерируем $u \sim U_{[0,1]}$. Если $u \leq \varrho(z)$, вернуть z. Иначе перейти к шагу 1.

1.5 Статистический оценивание параметров слабо зависимых авторегрессионных временных рядов при наличии цензурирования справа

Рассмотим еще один подход к оцениванию параметров модели авторгерессионного временного ряда, основанный на свойстве, что далеко отстоящие значения временного ряда, как правило слабозависимы. Идея была предложена в [16] для случая округленных данных. В данной работе этот подход был применен для цензурированных данных. Для начала приведем несколько важных теоретических фактов, которые будут использованы для построения оценок.

Пусть наблюдается случайная многомерная выборка $Y = \left\{y_t, t = \overline{1,T}\right\}$, $y_t \in \mathbb{R}^{p+1}$, из многомерного нормального закона распределения с нулевым вектором математического ожидания и ковариационной матрицей специального Теплицева вида [10]:

$$\mathcal{L}\{y_t\} = \mathcal{N}_{p+1}(0,\Sigma),\ \Sigma = \begin{pmatrix} \sigma_0 & \sigma_1 & \dots & \sigma_p \\ \sigma_1 & \sigma_0 & \dots & \sigma_{p-1} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_p & \sigma_{p-1} & \dots & \sigma_0 \end{pmatrix} \in \mathbb{R}^{(p+1)\times(p+1)}, \tag{1.53}$$

где T – объем выборки, $(p+1)$ – размерность элементов выборки ($p \in \mathbb{N}$). Размерность записана в таком виде для удобства записи формул в дальнейшем.

Будем считать, что элементы ковариационной матрицы σ_0, σ_1, $\dots$, σ_p неизвестные параметры, которые необходимо оценить. Построим функцию правдоподобия, учитывая формулу плотности многомерного нормального закона распределения:

$$L(\Sigma|Y) = \prod_{t=1}^{T} p(\Sigma|y_t) = \prod_{t=1}^{T} (2\pi)^{-\frac{(p+1)}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) =$$

$$= (2\pi)^{-\frac{(p+1)T}{2}} |\Sigma|^{-\frac{T}{2}} \prod_{t=1}^{T} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right). \tag{1.54}$$

Прологарифмировав (1.54) найдем логарифмическую функцию правдоподобия:

$$l(\Sigma|Y) = \ln L(\Sigma|Y) = -\frac{(p+1)T}{2}\ln(2\pi) - \frac{T}{2}\ln|\Sigma| + \sum_{t=1}^{T}\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right). \quad (1.55)$$

Лемма 1.1. *Пусть матрица* Σ *имеет специальный вид (1.53), тогда выполнено следующее соотношение:*

$$\frac{\partial}{\partial\sigma_i}\Sigma^{-1} = -\Sigma^{-1}I_i\Sigma^{-1}, \quad (1.56)$$

где $(I_i)_{k,l} = \delta_{i,|k-l|}$, $i,k,l \in \{0,\dots,p\}$, δ *– символ Кронекера.*

Доказательство. Согласно определению обратной матрицы $\Sigma^{-1}\times\Sigma = E_{p+1}$. Поскольку частная производная по σ_i от константной единичной матрицы всегда равно нулевой матрице, то получаем:

$$\frac{\partial}{\partial\sigma_i}\left(\Sigma^{-1}\Sigma\right) = \frac{\partial}{\partial\sigma_i}\Sigma^{-1}\Sigma + \Sigma^{-1}\frac{\partial}{\partial\sigma_i}\Sigma = 0_{p+1}. \quad (1.57)$$

Непосредственной проверкой убеждаемся, что $\frac{\partial}{\partial\sigma_i}\Sigma = I_i$, тогда подставив это значение в (1.57), получим:

$$\frac{\partial}{\partial\sigma_i}\Sigma^{-1}\Sigma = -\Sigma^{-1}I_i.$$

$$\frac{\partial}{\partial\sigma_i}\Sigma^{-1} = -\Sigma^{1}I_i\Sigma^{-1}.$$

□

Лемма 1.2. *Пусть* $A = (f_{i,j}(x))_{i,j=0}^{p}$ *- квадратная матрица размерности* $(p+1)\times(p+1)$*, тогда верно следующее соотношение для частной производной от определителя матрицы A:*

$$\frac{\partial}{\partial x}|A| = \sum_{k=0}^{p}|A_k|,\ A_k = \begin{pmatrix} f_{0,0}(x) & f_{0,1}(x) & \dots & f_{0,p}(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{k,0}(x)}{\partial x} & \frac{\partial f_{k,1}(x)}{\partial x} & \dots & \frac{\partial f_{k,p}(x)}{\partial x} \\ \vdots & \vdots & \ddots & \vdots \\ f_{p,0}(x) & f_{p,1}(x) & \dots & f_{p,p}(x) \end{pmatrix}, k \in \{0,\dots,p\}. \quad (1.58)$$

Доказательство. Доказательство проведем по методу математической индук-

ции по p. Применив известную формулу для определителя матрицы размерности 2, для $p = 1$ получим:

$$\frac{\partial}{\partial x}\left|\begin{pmatrix} f_{0,0}(x) & f_{0,1}(x) \\ f_{1,0}(x) & f_{1,1}(x) \end{pmatrix}\right| = \frac{\partial}{\partial x}(f_{0,0}(x)f_{1,1}(x) - f_{0,1}(x)f_{1,0}(x)) =$$

$$= \frac{\partial f_{0,0}(x)}{\partial x}f_{1,1}(x) + f_{0,0}(x)\frac{\partial f_{1,1}(x)}{\partial x} - \frac{\partial f_{0,1}(x)}{\partial x}f_{1,0}(x) - f_{0,1}(x)\frac{\partial f_{1,0}(x)}{\partial x} =$$

$$= \left(\frac{\partial f_{0,0}(x)}{\partial x}f_{1,1}(x) - \frac{\partial f_{0,1}(x)}{\partial x}f_{1,0}(x)\right) + \left(f_{0,0}(x)\frac{\partial f_{1,1}(x)}{\partial x} - f_{0,1}(x)\frac{\partial f_{1,0}(x)}{\partial x}\right) =$$

$$= \left|\begin{pmatrix} \frac{\partial f_{0,0}(x)}{\partial x} & \frac{\partial f_{0,1}(x)}{\partial x} \\ f_{1,0}(x) & f_{1,1}(x) \end{pmatrix}\right| + \left|\begin{pmatrix} f_{0,0}(x) & f_{0,1}(x) \\ \frac{\partial f_{1,0}(x)}{\partial x} & \frac{\partial f_{1,1}(x)}{\partial x} \end{pmatrix}\right|.$$

Предположим, что соотношение (1.58) верно для матриц размерности $p \times p$. Распишем производную по x для определителя матрицы размерности $(p+1) \times (p+1)$ по p-ой строке:

$$\frac{\partial}{\partial x}|A| = \frac{\partial}{\partial x}\left(\sum_{j=0}^{p}(-1)^{p+j}\bar{A}_{p,j}f_{p,j}(x)\right),$$

где $\bar{A}_{p,j}$ – дополнительные миноры, поскольку при удалении столбца и строки из матрицы A получится матрица размерности $p \times p$, т.е. по предположению индукции для вычисления производных от дополнительных миноров можно воспользоваться формулой (1.58), получаем:

$$\frac{\partial}{\partial x}|A| = \sum_{j=0}^{p}(-1)^{p+j}\left(\frac{\partial}{\partial x}\bar{A}_{p,j}f_{p,j}(x) + \bar{A}_{p,j}\frac{\partial f_{p,j}(x)}{\partial x}\right) =$$

$$= \sum_{j=0}^{p}(-1)^{p+j}\frac{\partial}{\partial x}\bar{A}_{p,j}f_{p,j}(x) + \sum_{j=0}^{p}(-1)^{p+j}\bar{A}_{p,j}\frac{\partial f_{p,j}(x)}{\partial x} =$$

$$= \sum_{j=0}^{p}(-1)^{p+j}\sum_{k=0}^{p-1}\bar{A}^{k}_{p,j}f_{p,j}(x) + \sum_{j=0}^{p}(-1)^{p+j}\bar{A}_{p,j}\frac{\partial f_{p,j}(x)}{\partial x},$$

где

$$\bar{A}^k_{p,j} = \left| \begin{pmatrix} f_{0,0}(x) & \dots & f_{0,j-1}(x) & f_{0,j+1}(x) & \dots & f_{0,p}(x) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_{k,0}(x)}{\partial x} & \dots & \frac{\partial f_{k,j-1}(x)}{\partial x} & \frac{\partial f_{k,j+1}(x)}{\partial x} & \dots & \frac{\partial f_{k,p}(x)}{\partial x} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ f_{p-1,0}(x) & \dots & f_{p-1,j-1}(x) & f_{p-1,j+1}(x) & \dots & f_{p-1,p}(x) \end{pmatrix} \right|, j \in \{0, \dots, p\}.$$

Поменяв порядок суммирования в первом слагаемом, получим:

$$\frac{\partial}{\partial x}|A| = \sum_{k=0}^{p-1}\sum_{j=0}^{p}(-1)^{p+j}\bar{A}^k_{p,j}f_{p,j}(x) + \sum_{j=0}^{p}(-1)^{p+j}\bar{A}_{p,j}\frac{\partial f_{p,j}(x)}{\partial x} =$$

$$= \sum_{k=0}^{p-1}|A_k| + |A_p| = \sum_{k=0}^{p}|A_k|.$$

□

Замечание 1.2. В случае небольшой размерности ($p = 2, 3$) для вычисления частной производной от определителя матрицы, можно вместо применения (1.58) вычислить определитель в явном виде и взять от него требуемую производную.

Для построения оценок максимального правдоподобия функция правдоподобия может быть максимизирована методом градиентного спуска:

$$\begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \sigma_p \end{pmatrix}^{(k+1)} = \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \vdots \\ \sigma_p \end{pmatrix}^{(k)} + \alpha \operatorname{grad} l\left(\Sigma^k|Y\right), \tag{1.59}$$

где α – шаг спуска. Для вычисления градиента может быть применена следующая теорема или следствие из нее.

Теорема 1.3. *Пусть случайная выборка* $Y = \left\{y_t, t = \overline{1,T}\right\}$ *описывается моделью (1.53), тогда градиент от функции правдоподобия, как функции от* $(\sigma_0, ..., \sigma_p)$*, может быть вычислен по следующей формуле:*

$$\operatorname{grad} l\left(\Sigma|Y\right) = \left(-\frac{T}{2}|\Sigma|^{-1}\sum_{k=0}^{p}S_{i,k} + \sum_{t=1}^{T}\frac{1}{2}y_t^T\Sigma^{-1}I_i\Sigma^{-1}y_t,\ i \in \{0, \dots, p\}\right), \tag{1.60}$$

где $S_{i,j}$ это определитель следующего вида:

$$S_{i,k} = \begin{vmatrix} \sigma_0 & \sigma_1 & \dots & \sigma_p \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial\sigma_i}\sigma_j & \frac{\partial}{\partial\sigma_i}\sigma_{j-1} & \dots & \frac{\partial}{\partial\sigma_i}\sigma_{p-j} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_p & \sigma_{p-1} & \dots & \sigma_0 \end{vmatrix}, \; i,k \in \{0,\dots,p\}. \tag{1.61}$$

Доказательство. Вычислим частные производные по σ_i от логарифмической функции правдоподобия (1.55). Поскольку производная от суммы непрерывных функций равна сумме производных от каждого слагаемого, то вычислим частные производные от каждого слагаемого по-отдельности:

$$\frac{\partial}{\partial\sigma_i} - \frac{(p+1)T}{2}\ln(2\pi) = 0; \tag{1.62}$$

$$\frac{\partial}{\partial\sigma_i} - \frac{T}{2}\ln|\Sigma| = -\frac{T}{2}|\Sigma|^{-1}\frac{\partial}{\partial\sigma_i}|\Sigma| = -\frac{T}{2}|\Sigma|^{-1}\sum_{k=0}^{p} S_{i,k}; \tag{1.63}$$

$$\frac{\partial}{\partial\sigma_i}\sum_{t=1}^{T}\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right) = -\frac{1}{2}\sum_{t=1}^{T}\left(y_t^T\frac{\partial}{\partial\sigma_i}\Sigma^{-1}y_t\right) = -\frac{1}{2}\sum_{t=1}^{T}\left(-y_t^T\Sigma^{-1}I_i\Sigma^{-1}y_t\right). \tag{1.64}$$

Для вычисления (1.63) были использованы результаты леммы 1.2, для вычисления (1.64) были использованы результаты леммы 1.1. Сложив выражения (1.62) – (1.64), получим требуемый результат. □

Следствие 1.1. *При $p = 1$ градиент (1.60) примет следующий вид:*

$$\operatorname{grad} l\,(\Sigma|Y) = \begin{pmatrix} \frac{-T\sigma_0}{\sigma_0^2-\sigma_1^2} + \frac{1}{2(\sigma_0^2-\sigma_1^2)^2}\sum\limits_{t=1}^{T} y_t^T\begin{pmatrix}\sigma_0^2+\sigma_1^2 & -2\sigma_0\sigma_1\\ -2\sigma_0\sigma_1 & \sigma_0^2+\sigma_1^2\end{pmatrix}y_t, \\ \frac{T\sigma_1}{\sigma_0^2-\sigma_1^2} + \frac{1}{2(\sigma_0^2-\sigma_1^2)^2}\sum\limits_{t=1}^{T} y_t^T\begin{pmatrix}-2\sigma_0\sigma_1 & \sigma_0^2+\sigma_1^2\\ \sigma_0^2+\sigma_1^2 & -2\sigma_0\sigma_1\end{pmatrix}y_t \end{pmatrix}. \tag{1.65}$$

Доказательство. Запишем в явном виде градиент (1.60) при $p = 1$:

$$\operatorname{grad} l\,(\Sigma|X) = \begin{pmatrix} \frac{-T}{2}|\Sigma|^{-1}\frac{\partial}{\partial\sigma_1}|\Sigma| + \frac{1}{2}\sum\limits_{t=1}^{T} y_t^T\Sigma^{-1}I_0\Sigma^{-1}y_t, \\ \frac{-T}{2}|\Sigma|^{-1}\frac{\partial}{\partial\sigma_2}|\Sigma| + \frac{1}{2}\sum\limits_{t=1}^{T} y_t^T\Sigma^{-1}I_1\Sigma^{-1}y_t \end{pmatrix}. \tag{1.66}$$

Согласно (1.53) при $p = 1$ ковариационная матрицы Σ имеет вид:

$$\Sigma = \begin{pmatrix} \sigma_0 & \sigma_1 \\ \sigma_1 & \sigma_0 \end{pmatrix},$$

тогда $|\Sigma| = \sigma_0^2 - \sigma_1^2$. Согласно лемме 1.1 матрицы I_0 и I_1 имеют вид:

$$I_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, I_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Подставив в (1.66) найденные выше значения и вычислив производные от определителя матрицы Σ получим следующее:

$$\operatorname{grad} l\,(\Sigma|Y) = \begin{pmatrix} \frac{-T\sigma_0}{\sigma_0^2-\sigma_1^2} + \frac{1}{2}\sum\limits_{t=1}^{T} y_t^T \Sigma^{-2} y_t, \\ \frac{T\sigma_1}{\sigma_0^2-\sigma_1^2} + \frac{1}{2}\sum\limits_{t=1}^{T} y_t^T \Sigma^{-1} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \Sigma^{-1} y_t \end{pmatrix}. \tag{1.67}$$

Воспользовавшись известной формулой для нахождения матрицы обратной к матрице размерности 2×2, получим:

$$\Sigma^{-1} = \begin{pmatrix} \sigma_0 & \sigma_1 \\ \sigma_1 & \sigma_0 \end{pmatrix}^{-1} = \frac{1}{\sigma_0^2 - \sigma_1^2} \begin{pmatrix} \sigma_0 & -\sigma_1 \\ -\sigma_1 & \sigma_0 \end{pmatrix}.$$

Подставив полученную матрицу в (1.67), получим требуемый по условию следствия вид градиента. □

Пусть теперь наблюдаемый временной ряд подвержен цензурированию. Будем считать, что для каждого t, $(t \in \{1, \ldots, T\})$ известен набор компонент $C_t = (c_{t,1}, \ldots, c_{t,m_t})$ $(C_t \subseteq \{0, \ldots, p\})$ вектора y_t, которые были подвержены цензурированию, $c_{t,i} \in \{0, \ldots p\}$ – индекс i-й цензурированной компоненты наблюдения y_t, $(i \in \{1, \ldots, m_t\})$; $m_t \in \{0, \ldots, p+1\}$ – количество цензурированных компонент в наблюдении y_t. Цензурирование означает, что вместо истинного значения компоненты вектора $y_{t,c_{t,i}}$ наблюдается лишь случайное событие $y_{t,c_{t,i}} \in [a_{t,c_{t,i}}, b_{t,c_{t,i}})$. В случае цензурирования справа $b_{t,c_{t,i}} = +\infty$ для любого t и i.

В векторном виде условие цензурирования примет в следующем виде:

$$y_t = \begin{pmatrix} y_{O_t} \\ y_{C_t} \end{pmatrix},\ y_{O_t} \in \mathbb{R}^{p+1-m_t},\ y_{C_t} \in V_t = \otimes_{i=1}^{m_t} [a_{t,c_{t,i}}, b_{t,c_{t,i}}),\ O_t = \{0, \ldots p\} \setminus C_t, \tag{1.68}$$

где O_t – номера компонент вектора y_t, которые наблюдаются полностью.

В условии (1.68) функция правдоподобия примет вид:

$$L(\Sigma|Y) = \prod_{t=1}^{T} p(\Sigma|y_t) = \prod_{t=1}^{T} \int\limits_{V_t} (2\pi)^{-\frac{(p+1)}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) dy_{C_t} =$$

$$= (2\pi)^{-\frac{(p+1)T}{2}} |\Sigma|^{-\frac{T}{2}} \prod_{t=1}^{T} \int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) dy_{C_t}. \tag{1.69}$$

Прологарифмировав (1.69) найдем логарифмическую функцию правдоподобия:

$$l(\Sigma|Y) = -\frac{(p+1)T}{2} \ln(2\pi) - \frac{T}{2} \ln|\Sigma| + \sum_{t=1}^{T} \ln \int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) dy_{C_t}. \tag{1.70}$$

Как и в случае отсутствия цензурирования, оценки максимального правдоподобия могут быть построены методом градиентного спуска [5]. Для вычисления градиента от функции правдоподобия может быть применена следующая теорема или следствие из нее.

Теорема 1.4. *Пусть случайная выборка* $Y = \left\{y_t, t = \overline{1,T}\right\}$ *описывается моделью (1.53) и подвержена цензурированию (1.68), тогда градиент от функции правдоподобия, как функции от* $(\sigma_0, ..., \sigma_p)$*, может быть вычислен по следующей формуле:*

$$\operatorname{grad} l\,(\Sigma|Y) = \left(-\frac{T}{2} |\Sigma|^{-1} \sum_{k=0}^{p} S_{i,k} + \sum_{t=1}^{T} \frac{\int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) y_t^T \Sigma^{-1} I_i \Sigma^{-1} y_t dy_{C_t}}{2 \int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) dy_{C_t}} = 0, \right. \tag{1.71}$$

$$\left. i \in \{0, \ldots, p\} \right),$$

где $S_{i,j}$ *определяются по формуле (1.61).*

Доказательство. Доказательство проведем аналогично доказательству теоремы 1.3. Поскольку первые два слагаемых логарифмической функции правдоподобия в случае цензурирования (1.70) и в случае полных данных (1.55) совпадают, то для нахождения частных производных по σ_i можно воспользоваться

готовыми результатами (1.62), (1.63) из доказательства теоремы 1.3. Остается вычислить частную производную от третьего слагаемого выражения (1.70):

$$\frac{\partial}{\partial\sigma_i}\sum_{t=1}^{T}\ln\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}=\sum_{t=1}^{T}\frac{\partial}{\partial\sigma_i}\ln\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}=$$

$$=\sum_{t=1}^{T}\frac{\frac{\partial}{\partial\sigma_i}\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}}{\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}}=\sum_{t=1}^{T}\frac{\int\limits_{V_t}\frac{\partial}{\partial\sigma_i}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}}{\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}}=$$

$$=\sum_{t=1}^{T}\frac{\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)\frac{\partial}{\partial\sigma_i}\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}}{\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}}=$$

$$=\sum_{t=1}^{T}\frac{\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)y_t^T\Sigma^{-1}I_i\Sigma^{-1}y_t dy_{C_t}}{2\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}}. \tag{1.72}$$

Сложив (1.62), (1.63), (1.72), получим требуемый градиент. □

Замечание 1.3. В случае, если для некоторого t будет выполнено $m_t = 0$, т.е. у наблюдения все компоненты будут известны точно, то дробь во втором слагаемом в формуле (1.71) примет следующий вид:

$$\frac{\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)y_t^T\Sigma^{-1}I_i\Sigma^{-1}y_t dy_{C_t}}{2\int\limits_{V_t}\exp\left(-\frac{1}{2}y_t^T\Sigma^{-1}y_t\right)dy_{C_t}}=\frac{1}{2}y_t^T\Sigma^{-1}I_i\Sigma^{-1}y_t,$$

что согласуется с видом слагаемых в формуле (1.60) для случая отсутствия цензурирования.

Следствие 1.2. *При $p = 1$ градиент (1.71) примет следующий вид:*

$$\operatorname{grad} l\,(\Sigma|Y) = \begin{pmatrix} A_1 + A_3 \sum\limits_{t=1}^{T} \dfrac{\int\limits_{V_t} \exp\left(\frac{-1}{2(\sigma_0^2-\sigma_1^2)} y_t^T \begin{pmatrix} \sigma_0 & -\sigma_1 \\ -\sigma_1 & \sigma_0 \end{pmatrix} y_t\right) y_t^T \begin{pmatrix} \sigma_0^2+\sigma_1^2 & -2\sigma_0\sigma_1 \\ -2\sigma_0\sigma_1 & \sigma_0^2+\sigma_1^2 \end{pmatrix} y_t dy_{C_t}}{\int\limits_{V_t} \exp\left(\frac{-1}{2(\sigma_0^2-\sigma_1^2)} y_t^T \begin{pmatrix} \sigma_0 & -\sigma_1 \\ -\sigma_1 & \sigma_0 \end{pmatrix} y_t\right) dy_{C_t}}, \\ A_2 + A_3 \sum\limits_{t=1}^{T} \dfrac{\int\limits_{V_t} \exp\left(\frac{-1}{2(\sigma_0^2-\sigma_1^2)} y_t^T \begin{pmatrix} \sigma_0 & -\sigma_1 \\ -\sigma_1 & \sigma_0 \end{pmatrix} y_t\right) y_t^T \begin{pmatrix} -2\sigma_0\sigma_1 & \sigma_0^2+\sigma_1^2 \\ \sigma_0^2+\sigma_1^2 & -2\sigma_0\sigma_1 \end{pmatrix} y_t dy_{C_t}}{\int\limits_{V_t} \exp\left(\frac{-1}{2(\sigma_0^2-\sigma_1^2)} y_t^T \begin{pmatrix} \sigma_0 & -\sigma_1 \\ -\sigma_1 & \sigma_0 \end{pmatrix} y_t\right) dy_{C_t}} \end{pmatrix}, \tag{1.73}$$

где $A_1 = \frac{-T\sigma_0}{\sigma_0^2-\sigma_1^2}$, $A_2 = \frac{T\sigma_1}{\sigma_0^2-\sigma_1^2}$, $A_3 = \frac{1}{2(\sigma_0^2-\sigma_1^2)^2}$.

Доказательство. Запишем градиент в явном виде (1.71) при $p = 1$:

$$\operatorname{grad} l\,(\Sigma|Y) = \begin{pmatrix} \frac{-T}{2}|\Sigma|^{-1}\frac{\partial}{\partial\sigma_1}|\Sigma| + \sum\limits_{t=1}^{T} \dfrac{\int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) y_t^T \Sigma^{-1} I_1 \Sigma^{-1} y_t dy_{C_t}}{2\int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) dy_{C_t}}, \\ \frac{-T}{2}|\Sigma|^{-1}\frac{\partial}{\partial\sigma_2}|\Sigma| + \sum\limits_{t=1}^{T} \dfrac{\int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) y_t^T \Sigma^{-1} I_2 \Sigma^{-1} y_t dy_{C_t}}{2\int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) dy_{C_t}} \end{pmatrix}. \tag{1.74}$$

Как уже было показано при доказательстве следствия 1.1:

$$\Sigma = \begin{pmatrix} \sigma_0 & \sigma_1 \\ \sigma_1 & \sigma_0 \end{pmatrix},\ |\Sigma| = \sigma_0^2 - \sigma_1^2,\ I_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ I_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

Подставив эти значения в (1.74), получим следующую систему:

$$\operatorname{grad} l\,(\Sigma|Y) = \begin{pmatrix} \dfrac{-T\sigma_0}{\sigma_0^2-\sigma_1^2} + \frac{1}{2}\sum\limits_{t=1}^{T} \dfrac{\int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) y_t^T \Sigma^{-2} y_t dy_{C_t}}{\int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) dy_{C_t}}, \\ \dfrac{T\sigma_1}{\sigma_0^2-\sigma_1^2} + \frac{1}{2}\sum\limits_{t=1}^{T} \dfrac{\int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) y_t^T \Sigma^{-1} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \Sigma^{-1} y_t dy_{C_t}}{\int\limits_{V_t} \exp\left(-\frac{1}{2} y_t^T \Sigma^{-1} y_t\right) dy_{C_t}} \end{pmatrix}.$$

Подставив в нее выражения для Σ^{-1}, получим окончательный градиент. □

В модели авторегрессионных временных рядов AR(p) ковариация элементов быстро стремится к нулю [16]. Основываясь на этом, сделаем допущение,

что элементы временного ряда далеко отстоящие друг от друга – независимы.

Пусть наблюдается реализация стационарного авторегрессионного временного ряда порядка p (1.1). Разобьем исходный временной ряд $X = \{x_t \in \mathbb{R}, t = \overline{1,N}\}$ длины N на k подвыборок $(p+1)$-мерных векторов, чтобы длина каждой подвыборки была T:

$$X^{(s)} = \{x_t^{(s)}, t = \overline{1,T}\},\ s = \overline{1,k},\ x_t^{(s)} = \left(x_{s+(t-1)k}, x_{s+(t-1)k+1}, \ldots, x_{s+(t-1)k+p}\right)^T. \tag{1.75}$$

Найдем ограничения на параметры k и T описанной выше процедуры деления исходного временного ряда на подвыборки. Во-первых, вектора в одной подвыборке не должны пересекаться (не должны содержать одинаковые элементы в соседних векторах), а для этого необходимо, чтобы $k > p$, в противном случае вектора будут явно зависимыми. При этом, чем больше k будет выбрано, тем менее зависимы между собой будут вектора в одной подвыборке. Во-вторых, длина каждой подвыборки должна быть $T \geq p+1$, а лучше $T >> p+1$, чтобы по элементам подвыборки мы могли достаточно точно оценить искомые параметры. В-третьих, индекс последнего элемента последней подвыборки должен быть меньше N, чтобы все подвыборки могли быть построены из элементов временного ряда, т.е. $k + (T-1)k + p \leq N$, что равносильно $Tk \leq N - p$. Чтобы, в целом, описанная процедура была осуществима, учитывая оценки снизу на k и T, необходимо, чтобы было выполнено следующее неравенство $(p+1)^2 + p \leq N$. Если оно выполнено, то в качестве T и k можно выбрать следующие значения $T = k = \left[\sqrt{N-p}\right]$, где квадратные скобки означают нахождение целой части.

Если предположить, что векторы подвыборки $X^{(s)}$ независимы, то известно, что $\mathbf{E}\{x_t^{(x)}\} = 0$, а в силу стационарности ковариационная матрица имеет Теплицев вид (1.53), так как $\Sigma = (\sigma_{i,j}) = (\sigma_{|i-j|})$. Тогда для нахождения оценок ковариационной функции σ_i можно воспользоваться теоремой 1.3 в случае полных данных, либо 1.4 в случае наличия цензурирования. Имея оценки σ_i из системы уравнений Юла-Уокера можно найти оценки исходных параметров модели $\theta^{(s)}$. Указанную процедуру можно проделать для каждой подвыборки $X^{(s)}$ и получить набор оценок исходного временного ряда $\hat{\theta}^{(s)},\ s = \overline{1,k}$. Тогда окончательные оценки можно построить, как $\hat{\theta} = \frac{1}{k}\sum\limits_{s=1}^{k}\hat{\theta}^{(s)}$.

Исходя из изложенного выше, предлагаем следующий алгоритм построения оценок исходного временного ряда.

1. Выбираем параметры k, T, удовлетворяющие ограничениям: $k \geq p+1, T \geq p+1, Tk \leq N-p$.

2. Разбиваем исходный временной ряд на k подвыборок $(p+1)$-мерных век-

торов $X^{(s)}$ длины T согласно (1.75).

3. Для каждой подвыборки максимизируем функцию правдоподобия методом градиентного спуска (1.59) и находим оценки ковариационной функции $\hat{\sigma}_i^s,\ i \in \{0, \ldots, p\}, s \in \{1, \ldots, k\}$.

4. Для каждого набора оценок ковариационной функции, решая систему уравнений Юла-Уокера, найдем оценки параметров исходного авторегрессионного временного ряда $\hat{\theta}^{(s)},\ s \in \{1, \ldots, k\}$.

5. Окончательные оценки параметров исходного авторегрессионного временного ряда строятся по формуле $\hat{\theta} = \frac{1}{k} \sum\limits_{s=1}^{k} \hat{\theta}^{(s)}$.

Замечание 1.4. При применении метода градиентного спуска (1.59) необходимо строить начальное приближение. Для этого используем метод комплектных данных [7], который в случае случайних пропусков даёт состоятельные результаты.

$$\sigma_0 = \frac{\hat{\sigma}_{1,1} + \hat{\sigma}_{2,2}}{2};\ \sigma_1 = \hat{\sigma}_{1,2} = \hat{\sigma}_{2,1}; \tag{1.76}$$

$$\hat{\sigma}_{1,1} = \frac{\sum\limits_{t=1}^{T} x_{t,1}^2 \mathrm{O}_{x_{t,1}}}{\sum\limits_{t=1}^{T} \mathrm{O}_{x_{t,1}}}; \tag{1.77}$$

$$\hat{\sigma}_{1,2} = \hat{\sigma}_{2,1} = \frac{\sum\limits_{t=1}^{T} x_{t,1} x_{t,2} \mathrm{O}_{x_{t,1}} \mathrm{O}_{x_{t,2}}}{\sum\limits_{t=1}^{T} \mathrm{O}_{x_{t,1}} \mathrm{O}_{x_{t,2}}}; \tag{1.78}$$

$$\hat{\sigma}_{2,2} = \frac{\sum\limits_{t=1}^{T} x_{t,2}^2 \mathrm{O}_{x_{t,2}}}{\sum\limits_{t=1}^{T} \mathrm{O}_{x_{t,1}}}, \tag{1.79}$$

где $O_{x_{t,i}} = \begin{cases} 1, & \text{если } x_{t,i} \text{ наблюдается,} \\ 0, & \text{если } x_{t,i} \text{ не наблюдается.} \end{cases}$

1.6 Метод моментов

Рассмотрим еще один метод оценивания параметров модели авторегрессии, который не учитывает механизм порождения пропусков [8]. Как показано

в литературе [8, 9] в случае случайных пропусков данные оценки являются состоятельными и несмещенными.

Математическая модель (1.4) остаётся прежней, только вместо цензурированных наблюдений в ней рассматриваются «пропуски».

Шаблоном «пропусков» назовём последовательность двоичных векторов $O_t = \begin{cases} 1, & \text{если } x_t \text{ наблюдается,} \\ 0, & \text{если } x_t \text{ не наблюдается.} \end{cases}$

Обозначим минимальный и максимальный моменты времени с наблюдаемыми компонентами $t_- = 1$, $t_+ = T$.

Обозначим ковариации: $G_k = Cov\{x_{k+1}, x_1\} \in \mathbb{R}$, $g_{t-t',k,k'} = Cov\{x_{t+k}x_t, x_{t'+k'}x_{t'}\}$, $t, t^{'} \in \mathbb{Z}$, $k, k^{'} \in \{0, 1\}$. Назовём

$$T_0 = inf\{T^{'} \in \mathbb{N} : min_{k\in\{0,1\}} \sum_{t=1}^{T^{'}-k} O_{t+k}O_t > 0\}$$

критическим временем для заданного шаблона O_t. Для $T \geqslant T_0$ определим функции от шаблона ($\tau \in \mathbb{Z}, k, k^{'} \in \{0, 1\}$) :

$$c_{\tau,k,k'}(T) = \frac{T \sum\limits_{t,t'=1}^{T-1} O_{t+k}O_tO_{t'+k'}O_{t'}\delta_{t-t',\tau}}{\sum\limits_{t=1}^{T-k} O_{t+k}O_t \sum\limits_{t=1}^{T-k'} O_{t+k'}O_t},$$

выборочные ковариации G_k:

$$G_0 = \frac{\sum\limits_{t=1}^{T} x_t^2 O_t}{\sum\limits_{t=1}^{T} O_t}; \; G_1 = \frac{\sum\limits_{t=1}^{T-1} x_t x_{t+1} O_t O_{t+1}}{\sum\limits_{t=1}^{T-1} O_t O_{t+1}}, \tag{1.80}$$

и при $|G| \neq 0$ статистики, основанные на выборочных ковариациях (1.80):

$$\hat{a} = \frac{G_1}{G_0}, \hat{\sigma} = G_0 - \frac{G_1^2}{G_0}. \tag{1.81}$$

Теорема 1.5. [8] *Статистики (1.80) являются несмещенными оценками матриц G_k, $k \in \{0, 1\}$: $\mathbf{E}\{\hat{G}_k\} = G_k$ для выбранной модели. Если к тому же выполнены условия:*

1. момент четвёртого порядка u_t равномерно ограничен $|\mathbf{E}\{u_t^4\}| \leq Const$,

где $t \in \mathbb{Z}$*;*

2. *при* $T \to \infty$ *число наблюдаемых пар компонент векторов в один и тот же и в соседние моменты времени бесконечно увеличивается:* $\sum_{t=1}^{T-k} O_{t+k} O_t \to \infty$, $k \in \{0, 1\}$,

то при $T \to \infty$ *оценки (1.80) состоятельны в среднем квадратическом:* $\hat{G}_k \xrightarrow{\mathbf{L_2}} G_k$*, а статистики (1.81) являются состоятельными (по вероятности) оценками параметров* a, σ*:* $\hat{a} \xrightarrow{\mathbf{P}} a$*,* $\hat{\sigma} \xrightarrow{\mathbf{P}} \sigma$*.*

Теорема 1.5 позволяет утверждать, что при достаточно общих ограничениях на случайные пропуски, оценки (1.81) являются состоятельными оценками параметров модели (1.4).

1.7 Выводы

В данной главе были получены следующие основные результаты:

1. Приведены основные теоретические сведения, необходимые для понимания основных результатов работы.

2. Рассмотрен предложенный в литературе “метод вставки” [12], для реализации которого в случае авторегрессионного временного ряда была найдена обратная ковариационная матрица в общем виде и рассмотрен ряд алгоритмов генерации псевдослучайной величины из усеченного нормального закона.

3. Предложен ряд модификаций метода вставки, которые, как будет показано в следующей главе, при незначительной потере точности позволяют значительно повысить скорость построения оценок. Для всех предложенных модификаций были найдены формулы для условного математического ожидания и ковариационной матрицы, которые необходимы для реализации этих методов.

4. По аналогии с [16] были построены приближённые оценки максимального правдоподобия в случае цензурирования справа, найдены формулы для вычисления градиента.

5. Рассмотрены оценки, построенные по методу моментов, не учитывающие механизм порождения пропусков.

2 КОМПЬЮТЕРНЫЕ ЭКСПЕРИМЕНТЫ

2.1 Общее описание компьютерных экспериментов

Для исследования эмпирических свойств и проведения сравнительного анализа предложенных оценок, были проведены компьютерные эксперименты. Во всех экспериментах моделировался временной ряд авторегрессии порядка $p = 1$ с параметрами $T = 1000$, $a = 0,3$ или $a = -0,3, \sigma^2 = 1$, с уровнем цензурирования l. Затем проводилось цензурирование *справа* (сверху), т.е. все значения $x_t \geq l$ помечались, как пропущенные. Уровень цензурирования выбирался таким образом, чтобы доля цензурированных (пропущенных) значений временного ряда была равна некоторому наперед заданному числу. По полученному цензурированному сверху временному ряду строились оценки параметров модели авторегрессии (a, σ^2) методами, описанными в предыдущей главе.

Теоретически вычислить вариации оценок (1.2), рассмотренных в предыдущей главе, не удалось.

Для оценивания вариаций построенных оценок был использован метод Монте-Карло с числом итераций $M = 100$, тогда формулы для вычисления эмпирических вариаций имеют вид:

$$var(a) = \frac{1}{M} \sum_{i=1}^{M} (\hat{a}_i - a)^2 \tag{2.1}$$

и

$$var(\sigma^2) = \frac{1}{M} \sum_{i=1}^{M} (\hat{\sigma}_i^2 - \sigma^2)^2. \tag{2.2}$$

При изучении зависимости вариации оценок были рассмотрены следующие доли числа цензурированных наблюдений:

$$proportion \in \{0.001, 0.005, 0.03, 0.1, 0.25\}. \tag{2.3}$$

2.2 Сравнительный анализ метода вставки и его модификаций

Первая серия экспериментов была посвящена проведению сравнительного анализа точности оценивания параметров “методом вставки” и его модификациями, предложенными в первой главе.

В ходе проведения экспериментов были построены графики зависимости эмпирической вариации оценок $\hat{a}$ и $\hat{\sigma}^2$ от доли числа цензурированных наблюдений для рассмотренных в работе модификаций метода вставки при $a = 0,3$

(Рисунок 2.1 и Рисунок 2.2) и при $a = -0,3$ (Рисунок 2.3 и Рисунок 2.4).

В случае модели авторегресссии первого порядка наилучшие результаты показали следующие методы, которые будут использованы в последующих экспериментах, в сравнении с другими методами: модификации “1 any before”, “1 any before and 1 after” и “all observations” (общий случай “метода вставки”).

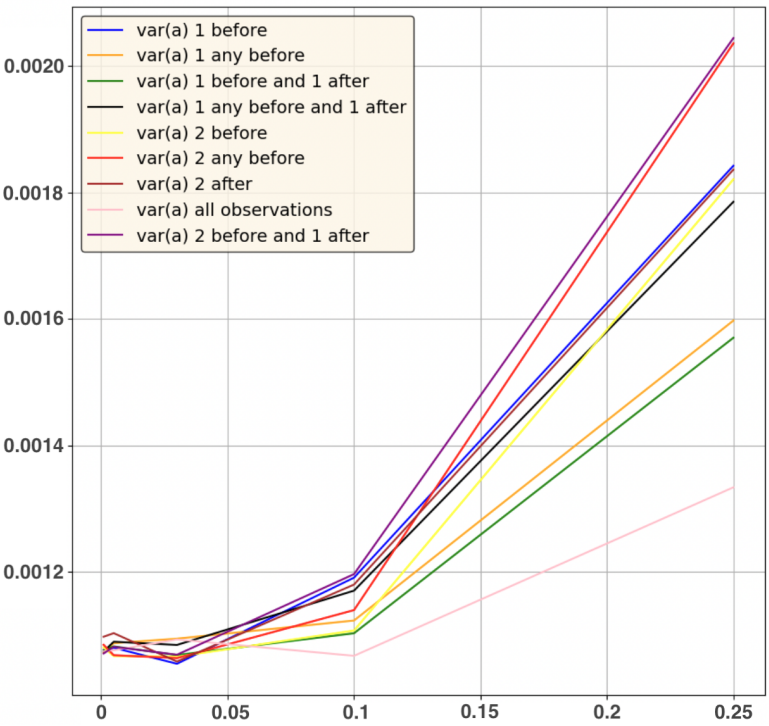

Рисунок 2.1: Эмпирическая вариация оценки $\hat{a}$ при $a = 0,3$

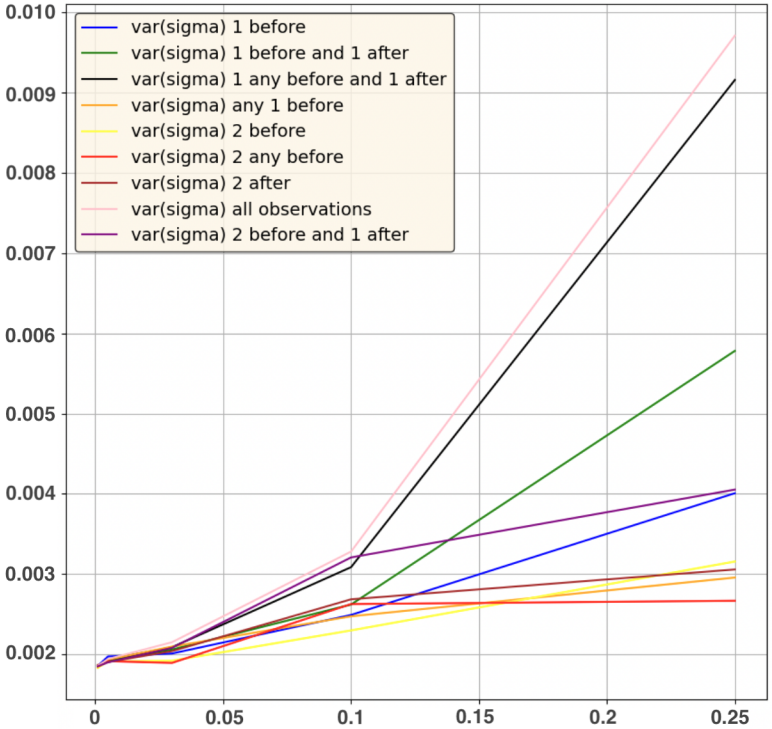

Рисунок 2.2: Эмпирическая вариация оценки $\hat{\sigma}^2$ при $a = 0,3$

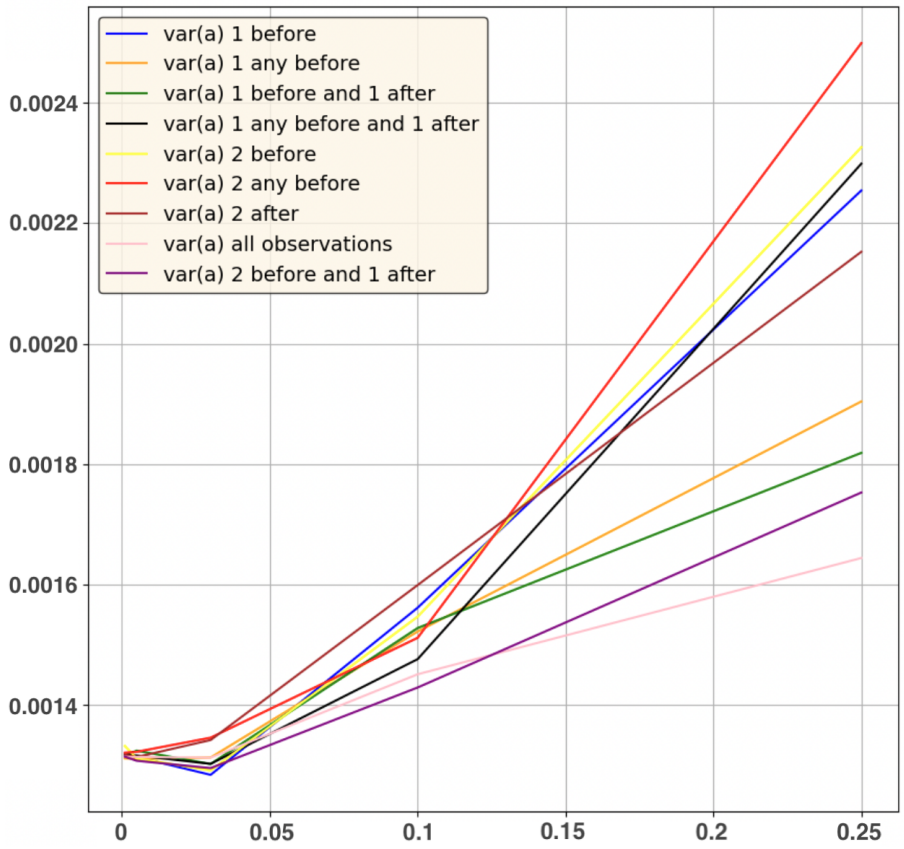

Рисунок 2.3: Эмпирическая вариация оценки $\hat{a}$ при $a = -0,3$

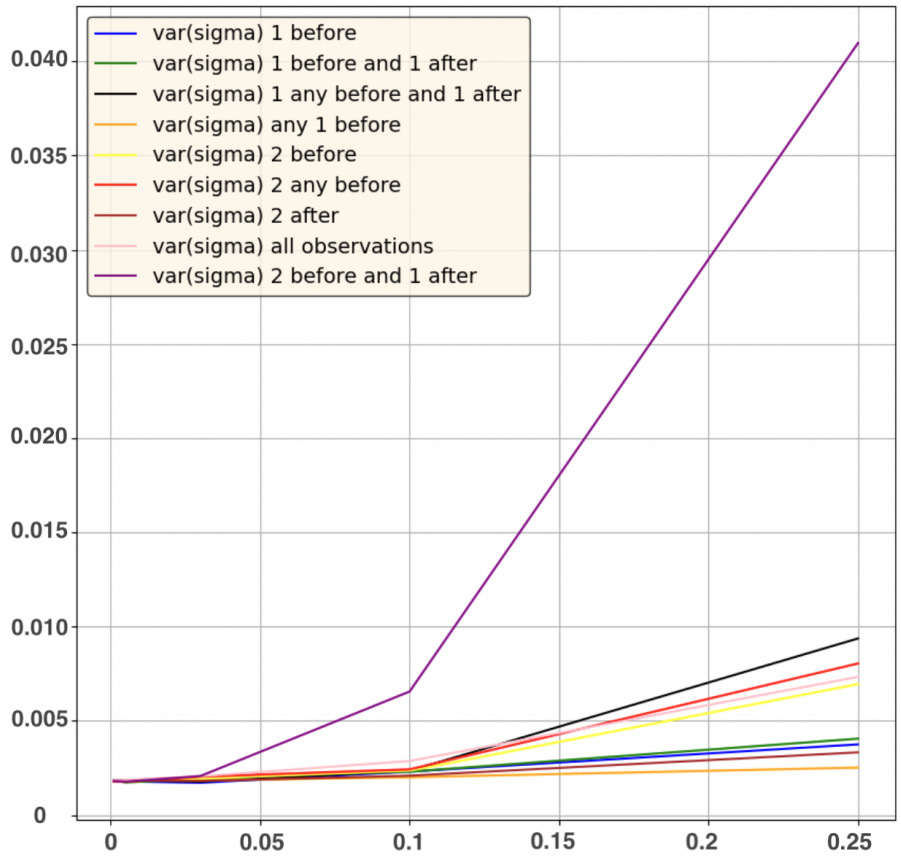

Рисунок 2.4: Эмпирическая вариация оценки $\hat{\sigma}^2$ при $a = -0,3$

Замечание 2.1. В общем случае авторегрессии порядка p, предполагается, что одной из наилучших модификаций с точки зрения точности оценивания будет "p any before".

2.3 Сравнительный анализ методов, учитывающих механизм порождения пропусков и не учитывающих

Был реализован метод моментов, который в случае случайных пропусков дает несмещённые и состоятельные оценки. Было проведено сравнение этого метода с другими методами, рассмотренными в данной работе, которые учитывают механизм появления пропусков. На (Рисунок 2.5) изображен график зависимости вариации оценок от доли цензурированных наблюдений. Как видно из рисунка, оценки, дающие хорошие результаты в случае случайных данных, могут давать неудовлетворительные результаты в случае цензурирования. Этот вывод согласуется с известными результатами в литературе [7].

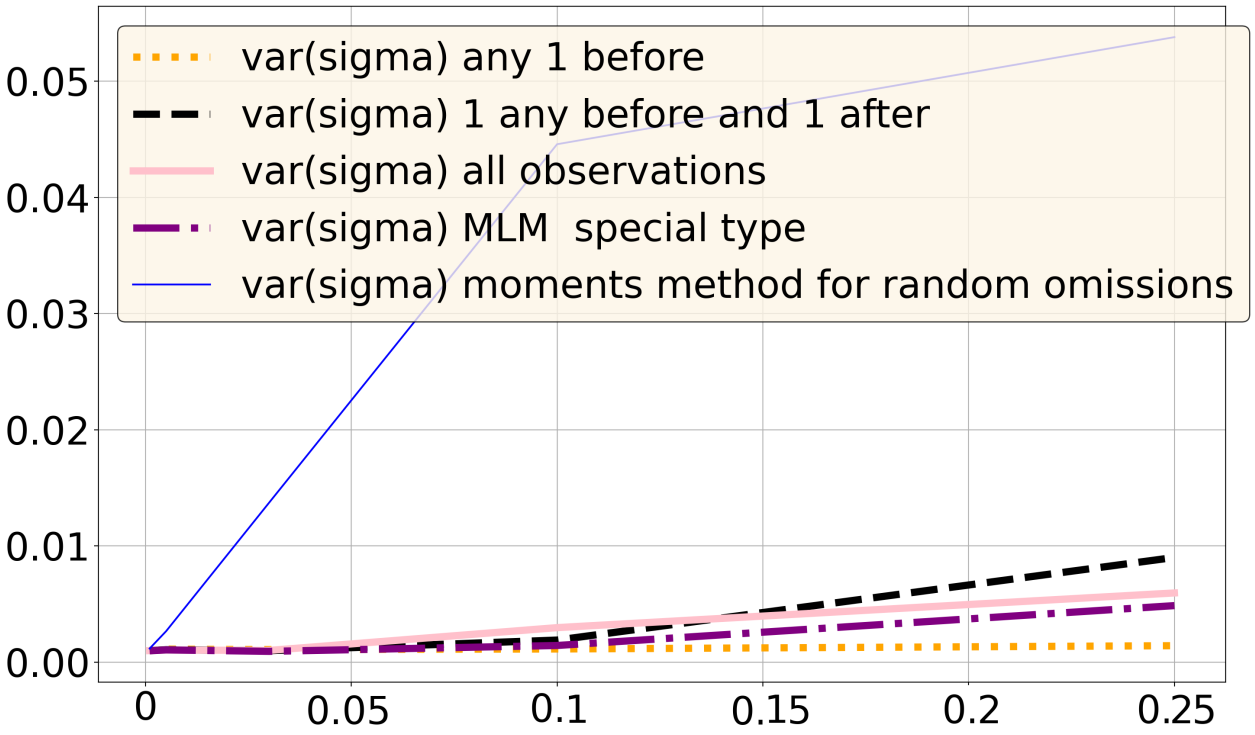

Рисунок 2.5: Вариация оценки $\hat{\sigma}^2$ при $a = 0,3$

2.4 Сравнительный анализ методов, учитывающих механизм цензурирования

Отдельно был проведён сравнительный анализ методов, учитывающий механиз порождения пропусков при цензурировании. В это сравнение был включен “метод вставки”, 2 выбранные в п.2.2 модификации “метода вставки” и приближённый метод максимального правдоподобия. В ходе компьютерных экспериментов были построены графики зависимости эмпирической вариации оценок $\hat{a}$ и $\hat{\sigma}^2$ от доли числа цензурированных наблюдений при $a = 0,3$ (Рисунок 2.6 и Рисунок 2.7) и при $a = -0,3$ (Рисунок 2.8 и Рисунок 2.9).

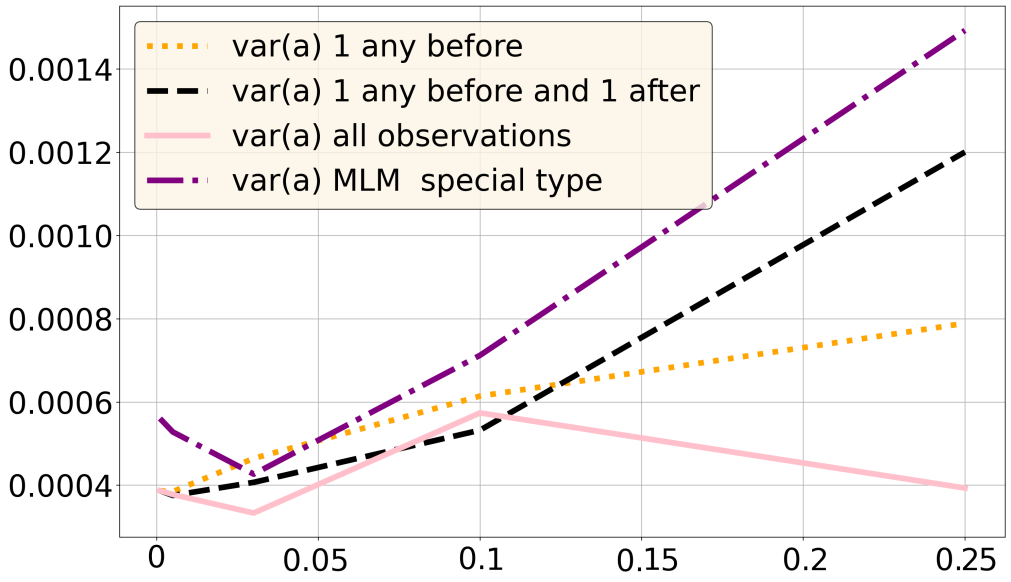

Рисунок 2.6: Эмпирическая вариация оценки $\hat{a}$ при $a = 0,3$

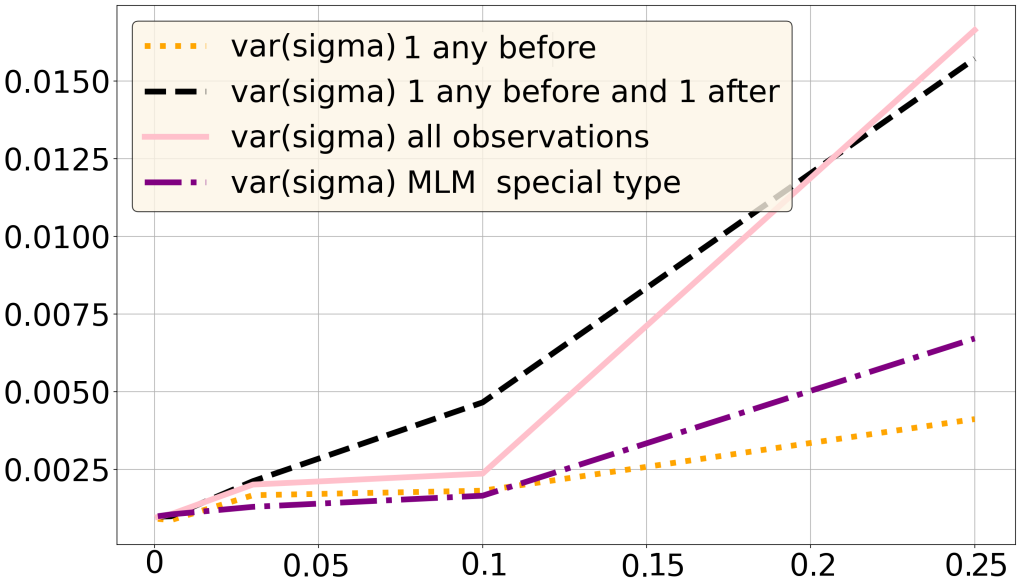

Рисунок 2.7: Эмпирическая вариация оценки $\hat{\sigma}^2$ при $a = 0,3$

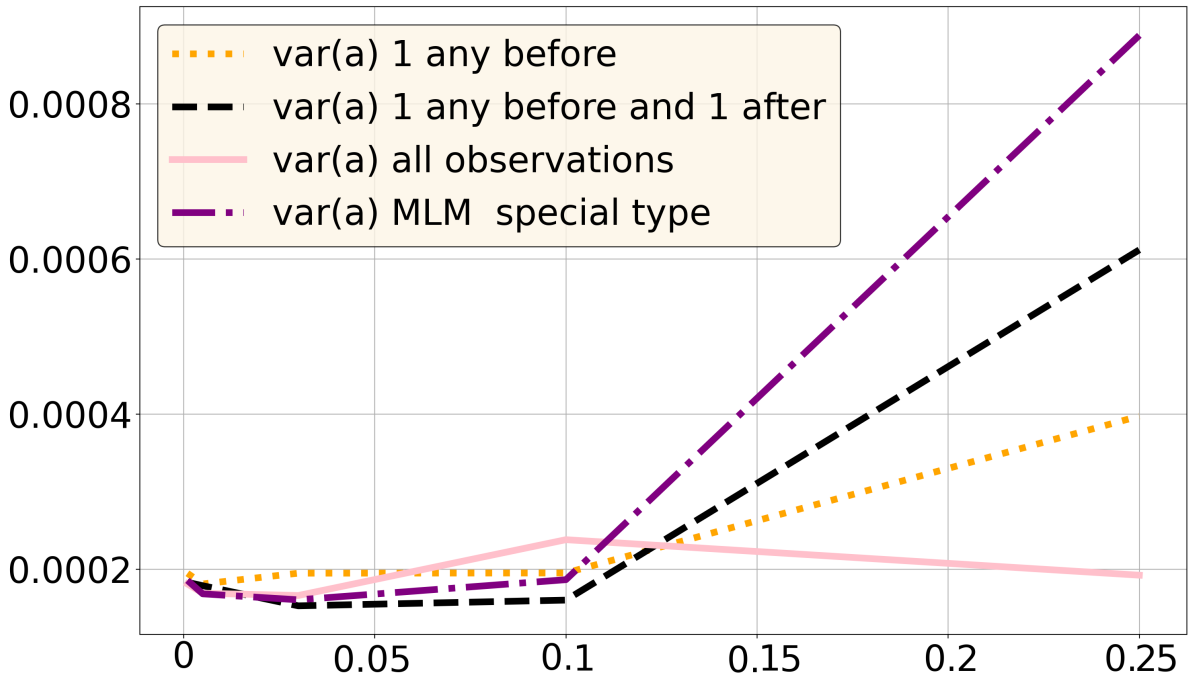

Рисунок 2.8: Эмпирическая вариация оценки $\hat{a}$ при $a = -0,3$

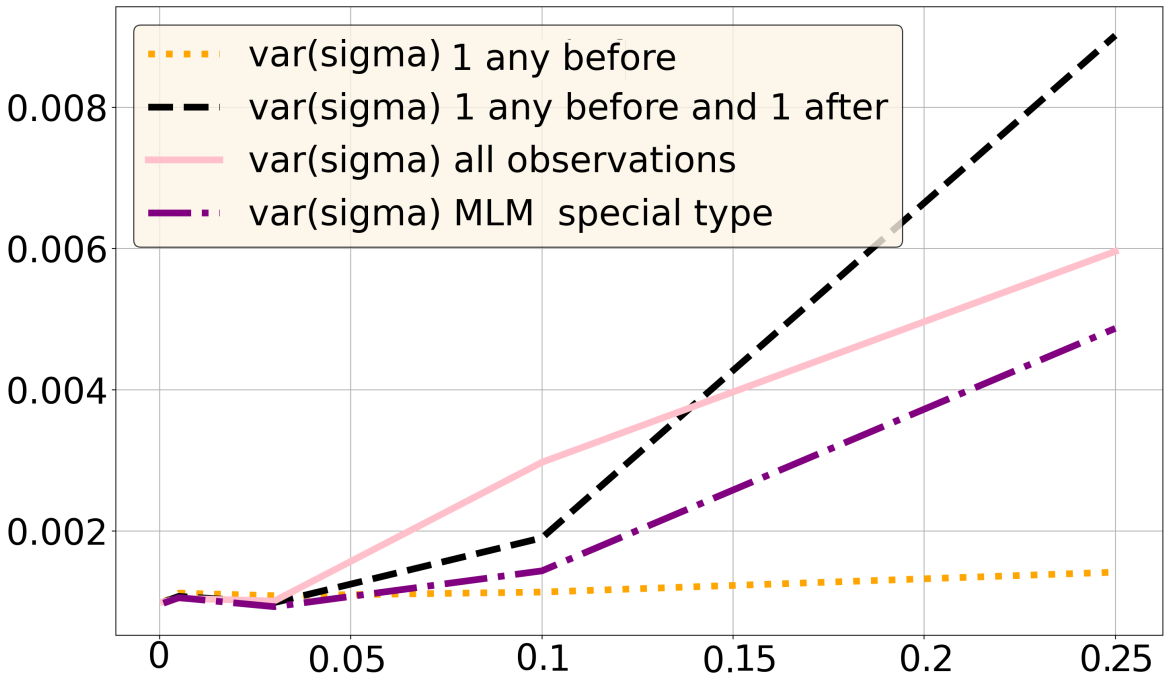

Рисунок 2.9: Эмпирическая вариация оценки $\hat{\sigma}^2$ при $a = -0,3$

На графиках (Рисунок 2.6 и Рисунок 2.7) и (Рисунок 2.8 и Рисунок 2.9) видно, что и для положительных a, и для отрицальных, метод “1 any before” показывает наилучшие по точности результаты, в особенности при оценивании дисперсии σ^2. Однако для оценивания параметра a лучше использовать “all observations” (общий случай “метода вставки”). Приближённый метод максимального правдоподобия (“MLM special type”) показывает одни из лучших результатов оценивания дисперсии σ^2, однако вместе с этим и худшие результаты при оценивании параметра a.

2.5 Сравнительный анализ реализованных методов по времени выполения

Проанализоровав рисунки (Рисунок 2.1 – Рисунок 2.4) можно сделать вывод, что в целом поведение эмпирической вариации оценок $\hat{a}$ и $\hat{\sigma}^2$ схоже для различных модификаций “метода вставки”. Расхождения можно объяснить накапливающимися погрешностями.

Из Рисунка 2.5 видно, что игнорирование механизма возникновения пропусков (в нашем случае цензурирования) приводит к существенному снижению точности вычислений.

При более детальном изучении методов из графиков (Рисунок 2.6 – Рисунок 2.9) можно сделать вывод, что модификация “1 any before” показывает наиболее точные результаты. Метод “all observations” (общий случай “метода вставки”) несколько точнее приближённого метода максимального правдоподобия.

В ходе выполенения компьютерных экспериментов было вычислено среднее время работы всех предложенных и реализованных в работе методов. Для “метода вставки” и его модификаций эти данные представлены в Таблицах 1 и 2 при $a = 0,3$ и $a = -0,3$ соответственно.

Таблица 1: Среднее время работы (в секундах) модификаций метода вставки при $a = 0,3$

	1 before	1 before and 1 after	2 before	2 after	2 any before	1 any before and 1 after	1 any before	all observations	2 before and 1 after
0.1	0.001238	0.002379	0.000892	0.001020	0.001004	0.004837	0.001433	0.588092	0.000741
0.5	0.002029	0.002560	0.001687	0.002148	0.001790	0.004724	0.002294	0.878005	0.001557
3.0	0.005905	0.005518	0.005042	0.005421	0.004815	0.008310	0.005230	2.437922	0.005025
10.0	0.010365	0.011447	0.012721	0.012950	0.011465	0.013905	0.010756	5.326676	0.016502
25.0	0.026932	0.028405	0.030084	0.025442	0.030837	0.031031	0.023551	10.376256	0.041819

Таблица 2: Среднее время работы (в секундах) модификаций метода вставки при $a = -0,3$

	1 before	1 before and 1 after	2 before	2 after	2 any before	1 any before and 1 after	1 any before	all observations	2 before and 1 after
0.1	0.001230	0.001140	0.001002	0.000964	0.000989	0.003963	0.001062	0.593369	0.000641
0.5	0.002374	0.002115	0.001847	0.002009	0.001839	0.004806	0.002393	0.922642	0.001525
3.0	0.004710	0.005671	0.004945	0.005332	0.005085	0.007977	0.005022	2.102931	0.006165
10.0	0.011464	0.010095	0.012146	0.013700	0.014237	0.012959	0.011896	4.689981	0.016239
25.0	0.029454	0.027560	0.033618	0.029111	0.038436	0.031821	0.025589	9.843819	0.061734

Из таблиц 1 и 2 видно, что метод “all observations” (общий случай “метода вставки”) сильно проигрывает в скорости любой из модификаций, как и предполагалось ранее. Рост времени работы всех модификаций в зависимости от числа цензурированных наблюдений близок к линейному.

Также было подсчитано среднее время работы приближённого метода максимального правдоподобия (“MLM special type”) и для сравнения были использованы модификации “1 any before”, “1 any before and 1 after” и “all observations” (общий случай “метода вставки”). Эти данные представленны в Таблицах 3 и 4 для $a = 0,3$ и $a = -0,3$ соответственно.

Таблица 3: Среднее время работы (в секундах) при $a = 0,3$

	1 any before	1 any before and 1 after	all observations	MLM special type
0.1	0.001697	0.001370	0.825272	0.436755
0.5	0.007550	0.002882	1.000062	2.027464
3.0	0.011571	0.007237	3.424063	24.441012
10.0	0.019273	0.011043	8.109243	142.455456
25.0	0.033757	0.044785	15.665323	623.802892

Таблица 4: Среднее время работы (в секундах) при $a = -0,3$

	1 any before	1 any before and 1 after	all observations	MLM special type
0.1	0.000678	0.001082	0.641859	0.304309
0.5	0.002953	0.002329	0.959919	0.906338
3.0	0.005046	0.007177	2.570611	4.159439
10.0	0.014552	0.010422	4.510311	21.421842
25.0	0.032390	0.029017	7.770826	175.581788

Как видно из таблиц 3 и 4, приближённый метод максимального правдоподобия сильно прогрывает в скорости и время выполнения быстро увеличивается. Это связано с особенностями максимизации функции методом градиентного спуска, а также необходимостью приближённо вычислять интегралы.

2.6 Исследование сходимости вариации оценок от объёма выборки

Теоретически доказать сходимость вариации оценок к нулю, как и состоятельность оценок, для предложенных в работе методов не удалось. Поэтому поведение вариации оценок было исследовано эмпирически.

На рисунках (Рисунок 2.10 и Рисунок 2.11) представлена зависимость эмперической вариации оценок параметров модели от длины наблюдаемого временного ряда.

На этих рисунках представлено поведение эмпирической вариации лишь для следующих методов: “all observations” и “1 any before”, а также для приближённых оценок максимального правдоподобия (“MLM special type”). Поведение вариации остальных модификаций “метода вставки” схоже с поведением вариации для метода “1 any before” и поэтому не было представлено. Эксперимент заключался в следующем: доля ценурированных наблюдений была зафиксирована на уровне 0.2, исходные параметры временного ряда были взяты $a = -0.3, \sigma^2 = 1$, эмпирическая вариации оценок вычислялась по методу Монте-Карло с числом итераций M = 100 для следующих длин временного ряда:

$$N_0 = 100; \quad N_i = N_0 + ih,$$

где N – количество значений выборки, $h = 100$ – шаг, $i \in \{0, ..., 9\}$.

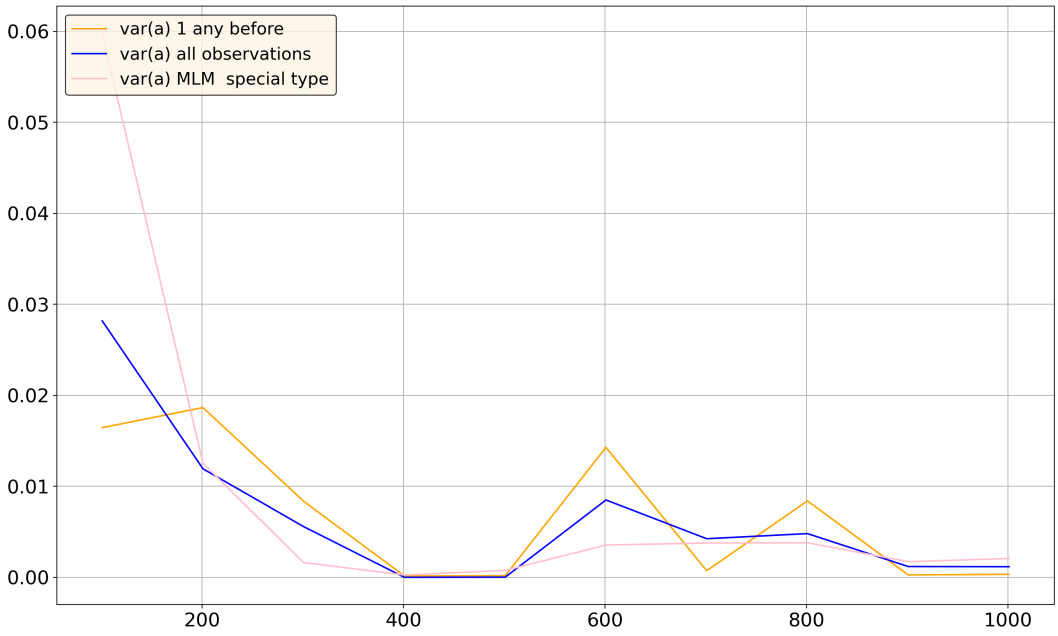

Рисунок 2.10: Вариация оценки $\hat{a}$ при $a = -0,3$ для выборок разной длины

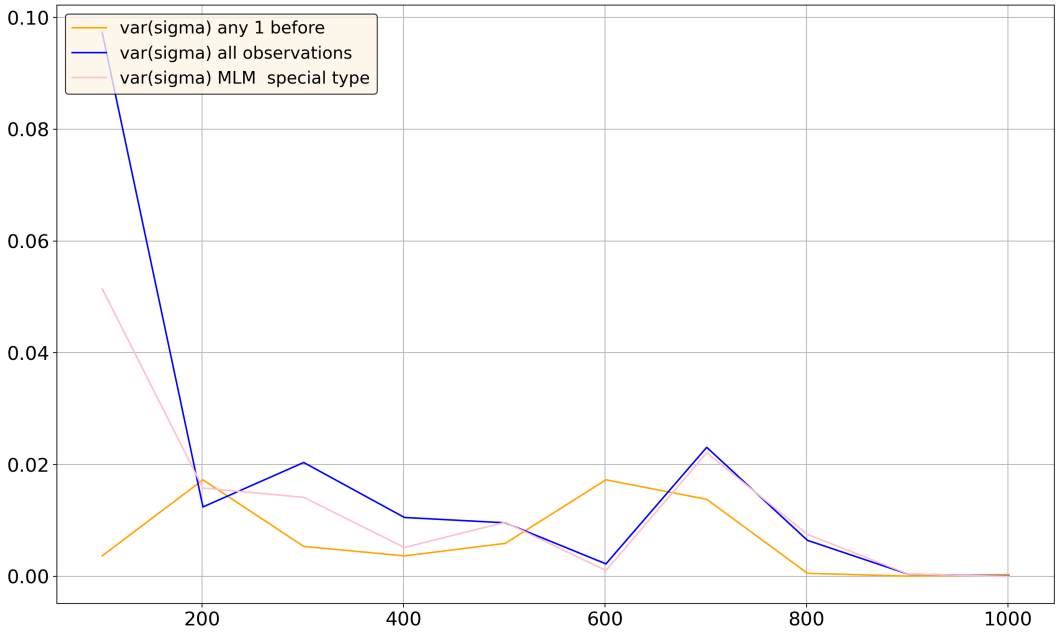

Рисунок 2.11: Вариация оценки $\hat{\sigma}^2$ при $a = -0,3$ для выборок разной длины

Из рисунков (Рисунок 2.10 – Рисунок 2.11) видно, что с увеличением длины выборки вариация уменьшается. Исходя из этого можно предположить, что предложенные оценки могут быть состоятельными.

2.7 Выводы

В данной главе были получены следующие основные результаты:

1. Проведён сравнительный анализ “метода вставки” и его модификаций, который показал, что модификации “1 any before”, “1 any before and 1 after” и “all observations” (общий случай “метода вставки”) дают наилучшие результаты по точности вычисления оценок среди всех рассмотренных вариантов “метода вставки”.

2. Проведён сравнительный анализ методов, учитывающих механизм порождения пропусков и не учитывающих, из которого следует: в случае цензурирования, оценки, дающие хорошие результаты в случае случайных данных, могут давать неудовлетворительные результаты, в связи с чем важно учитывать, как пропуски появились.

3. Проведён сравнительный анализ методов, учитывающих механизм порождения пропусков, по результатам которого модификация “1 any before” и “all observations” (общий случай “метода вставки”) показали наилучшие результаты среди всех методов, представленных в данной работе, как по точности, так и по скорости вычислений.

4. Проведён сравнительный анализ по времени выполнения, который показал что с ростом доли цензурированных значений время увеличивается. Приближённые оценки макисмального правдоподобия вычисляются существенно медленнее остальных.

5. В ходе проведения вычислительных экспериментов было показано, что эмпирическая вариация стремится к нулю с ростом длины наблюдаемого временного ряда. Это может свидетельствовать о том, что оценки являются состоятельными.

ЗАКЛЮЧЕНИЕ

В магистерской диссертации получены следующие основные результаты:

1. Рассмотрен предложенный в литературе “метод вставки”, для реализации которого в случае авторегрессионного временного ряда была найдена обратная ковариационная матрица в общем виде и рассмотрен ряд алгоритмов генерации псевдослучайной величины из усеченного нормального закона.

2. Предложен и реализован ряд модификаций “метода вставки”, которые, исходя из компьютерных экспериментов, без значительной потери точности позволяют значительно повысить скорость построения оценок по сравнению с оригинальным методом.

3. По аналогии с [16] были построены приближенные оценки максимального правдоподобия в случае цензурирования справа. Для максимизации приближенной функции правдоподобия использовался метод градиентного спуска, для применения которого, был найден градиент функции правдоподобия. Данный метод соизмерим по точности с “методом вставки” и его модификациями, но значительно уступает по времени работы.

4. Был реализован метод моментов статистического оценивания параметров авторегрессионого временного ряда при наличии случайных пропусков. Компьютерные эксперименты показали, что данный метод даёт неудовлетворительные результаты в случае цензурирования, когда наблюдения пропущены не случайно.

5. В ходе проведения вычислительных экспериментов было показано, что эмпирическая вариация стремится к нулю с ростом длины наблюдаемого временного ряда. Это может свидетельствовать о том, что оценки являются состоятельными.

Часть результатов магистерской диссертации докладывались на 13-ой международной научной конференции “Computer Data Analysis and Modeling” (2022), опубликованы материалы конференции [15].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Андерсон, Т. Статистический анализ временных рядов / Т. Андерсон – М.: Мир, 1976. – 755 с.

2. Бокс, Дж. Анализ временных рядов, прогноз и управление / Дж. Бокс, Г. Дженкинс; под ред. Писаренко В. Ф. – М.: Мир, 1974, кн. 1. – 406 с.

3. Бодягин, И. А. Функция правдоподобия для цензурированных гауссовских временных рядов / И. А. Бодягин, Ю. С. Харин // Весці нац. акад. навук. Беларусі. Сер. фіз.-мат. навук. — 2012. — № 2. — С. 4–14.

4. Ваттс, Д. Спектральный анализ и его приложения: в 2 т./ Д. Ваттс, Г. Дженкинс: МирГод, – 1971. – Т. 1. – 317 с.

5. Гасников А. В. Современные численные методы оптимизации. Метод универсального градиентного спуска : учебное пособие. — М.: МФТИ, 2018. — 291 с.

6. Криптология: учебник / Ю. С. Харин [и др.] – Минск: БГУ, 2013. – 511 с. – (Классическое университетское издание).

7. Литтл, Р.Дж.А. Статистический анализ данных с пропусками./ Р.Дж.А. Литтл, Д.Б. Рубин – М.: Финансы и статистика, 1990. – 336 с.

8. Харин, Ю.С. Методы прогнозирования векторных авторегрессионных временных рядов при наличии пропущенных значений / Ю.С. Харин, А.С. Гурин // Искусственный интеллект. – 2005. – №4. – С.292-301.

9. Харин, Ю.С. Оптимальность и робастность в статистическом прогнозировании / Ю. С. Харин – Минск: БГУ, 2008. – 263 с.

10. Харин, Ю. С. Теория вероятностей, математическая и прикладная статистика: учебник / Ю.С.Харин, Н.М.Зуев, Е.Е.Жук. – Минск: БГУ, 2011. – 464 с.

11. Эконометрия / В. И. Суслов [и др.] – Новосибирский государственный университет, 2005. – 744 с.

12. Park, J.W. Censored time series analysis with autoregressive moving average models / J.W. Park, M.G. Genton, S.K. Ghosh // The Canadian journal of statistics. – 2007. – Vol. 35, No. 1. – P. 151-168.

13. Press, William H. 2.9 Cholesky Decomposition / William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery // Numerical Recipes in C. — 2nd edition. — Cambridge: Cambridge University Press.

14. Robert, C.P. Simulation of truncated normal variables / C.P. Robert // Stat Comput. – 1995. – Vol. 5. – P. 121-125.

15. Tsirul, E.A. Modifications of the imputation method for parameter estimation of censored autoregressive time series / E.A. Tsirul, I.A. Badziahin // Proc. of the 13th Intern. Conf. «Computer Data Analysis and Modeling», Minsk. September, 6-10. – 2022. – P. 202–205.

16. Zhidong, B. Statistical analysis for rounded data / B. Zhidong, Z. Shurong, Z. Baoxue, Hu Guorong // Journal of Statistical Planning and Inference. – 2009 Vol. 139, No. 8. – P. 2526-2542.

ПРИЛОЖЕНИЕ А. Листинг программы метода вставки

```
def ins_w_1_obs(cs, est_a, est_sigma, level_c):
    timing = time.time()
    est_a_i = est_a
    est_sigma_i = est_sigma
    lte = 0
    cur_num_of_cen_elem = 0
    est_a_i1 = 1
    est_sigma_i1 = 2
    k = 0
    first_elem_exist = True

    while math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                                    est_sigma_i) >= eps:
        est_sample = []
        if k != 0:
            est_a_i = est_a_i1
            est_sigma_i = est_sigma_i1
        if cs[0] is np.nan:
            first_elem_exist = False
            temp = random.normalvariate(0,
            (est_sigma_i / (1 - est_a_i ** 2)) ** 0.5)
            while temp < level_c:
                temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
            cs[0] = temp
        for i in range(N):
            if cs[i] is not np.nan:
                lte = cs[i]
                cur_num_of_cen_elem = 0
                est_sample.append(lte)
            else:
                cur_num_of_cen_elem += 1
                mu_i = lte * est_a_i ** cur_num_of_cen_elem
                sigma_i = est_sigma_i * (1 - est_a_i **
                                         (2 * cur_num_of_cen_elem))
                norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                                    (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                while norm_var < level_c:
                    norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                                    (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                est_sample.append(norm_var)
        est_a_i1, est_sigma_i1 = est_coef_YW(est_sample)
        k += 1
```

```
        if not first_elem_exist:
            cs[0] = np.nan
    return [est_a_i1, est_sigma_i1, k, time.time() - timing]

def ins_w_1_obs_new(cs, est_a, est_sigma, level_c):
    timing = time.time()
    est_a_i = est_a
    est_sigma_i = est_sigma
    lte = 0
    est_a_i1 = 1
    est_sigma_i1 = 2
    k = 0
    first_elem_exist = True

    while math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                                    est_sigma_i) >= eps:
        est_sample = []
        if k != 0:
            est_a_i = est_a_i1
            est_sigma_i = est_sigma_i1
        if cs[0] is np.nan:
            first_elem_exist = False
            temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
            while temp < level_c:
                temp = random.normalvariate(0, (est_sigma_i /
                                    (1 - est_a_i ** 2)) ** 0.5)
            cs[0] = temp
        for i in range(N):
            if cs[i] is not np.nan:
                lte = cs[i]
                est_sample.append(lte)
            else:
                mu_i = lte * est_a_i
                sigma_i = est_sigma_i * (1 - est_a_i ** (2))
                norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                                (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                while norm_var < level_c:
                    norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                         (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                est_sample.append(norm_var)
                lte = norm_var
        est_a_i1, est_sigma_i1 = est_coef_YW(est_sample)
        k += 1
        if not first_elem_exist:
            cs[0] = np.nan
    return [est_a_i1, est_sigma_i1, k, time.time() - timing]
```

```
def ins_w_2_obs_bf(cs, est_a, est_sigma, level_c):
timing = time.time()
est_a_i = est_a
est_sigma_i = est_sigma
est_a_i1 = 1
est_sigma_i1 = 2
j = 0
cur_k = 0
l = 0
first_elem_ex = True
last_elem_ex = True

while math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                                est_sigma_i) >= eps:
    est_sample = []
    if j != 0:
        est_a_i = est_a_i1
        est_sigma_i = est_sigma_i1
    if cs[0] is np.nan:
        first_elem_ex = False
        temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
        while temp < level_c:
            temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
        cs[0] = temp
    if cs[N - 1] is np.nan:
        last_elem_ex = False
        temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
        while temp < level_c:
            temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
        cs[N - 1] = temp
    for index in range(N):
        if cs[index] is not np.nan:
            cur_k = 0
            l = 1
            est_sample.append(cs[index])
        else:
            cur_k += 1
            x_t = est_sample[index - cur_k]
            x_t_l = 0
            if cur_k == 1:
                while cs[index + l] is np.nan:
                    l += 1
```

```
                    x_t_l = cs[index + l]
                    l += 1
                mu_i = ((est_a_i ** cur_k - est_a_i ** (2 * l - cur_k)) *
                     x_t + (est_a_i ** (l - cur_k) - est_a_i ** (l +
                          cur_k)) * x_t_l) / (1 - est_a_i ** (2 * l))
                sigma_i = est_sigma_i * (1 - (est_a_i ** (2 * cur_k) +
                 est_a_i ** (2 * l - 2 * cur_k) - 2 * est_a_i **
                             (2 * l)) / ( 1 - est_a_i ** (2 * l)))
                norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                while norm_var < level_c:
                    norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                     (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                est_sample.append(norm_var)
        est_a_i1, est_sigma_i1 = est_coef_YW(est_sample)
        j += 1
        if not first_elem_ex:
            cs[0] = np.nan
        if not last_elem_ex:
            cs[N - 1] = np.nan
    return [est_a_i1, est_sigma_i1, j, time.time() - timing]

def ins_w_2_obs_bf_new(cs, est_a, est_sigma, level_c):
    timing = time.time()
    est_a_i = est_a
    est_sigma_i = est_sigma
    est_a_i1 = 1
    est_sigma_i1 = 2
    j = 0
    cur_k = 0
    l = 0
    first_elem_ex = True
    last_elem_ex = True

    while math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                                    est_sigma_i) >= eps:
        est_sample = []
        if j != 0:
            est_a_i = est_a_i1
            est_sigma_i = est_sigma_i1
        if cs[0] is np.nan:
            first_elem_ex = False
            temp = random.normalvariate(0, (est_sigma_i /
                                        (1 - est_a_i ** 2)) ** 0.5)
            while temp < level_c:
                temp = random.normalvariate(0, (est_sigma_i /
                                        (1 - est_a_i ** 2)) ** 0.5)
```

```
            cs[0] = temp
        if cs[N - 1] is np.nan:
            last_elem_ex = False
            temp = random.normalvariate(0, (est_sigma_i /
                                        (1 - est_a_i ** 2)) ** 0.5)
            while temp < level_c:
                temp = random.normalvariate(0, (est_sigma_i /
                                        (1 - est_a_i ** 2)) ** 0.5)
            cs[N - 1] = temp
        for index in range(N):
            if cs[index] is not np.nan:
                l = 1
                cur_k = 1
                est_sample.append(cs[index])
            else:
                x_t_k = est_sample[index - 1]
                if cur_k == 1:
                    while cs[index + l] is np.nan:
                        l += 1
                    x_t_l = cs[index + l]
                    l += 1
                mu_i = ((est_a_i ** cur_k - est_a_i ** (2 * l - cur_k)) *
                x_t + (est_a_i ** (l - cur_k) - est_a_i ** (l + cur_k)) *
                 x_t_l) / (1 - est_a_i ** (2 * l))
                sigma_i = est_sigma_i * (1 - (est_a_i ** (2 * cur_k) +
                 est_a_i ** (2 * l - 2 * cur_k) - 2 * est_a_i **
                 (2 * l)) / (1 - est_a_i ** (2 * l)))
                norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                 (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                while norm_var < level_c:
                    norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                     (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                est_sample.append(norm_var)
                l = 1
        est_a_i1, est_sigma_i1 = est_coef_YW(est_sample)
        j += 1
        if not first_elem_ex:
            cs[0] = np.nan
        if not last_elem_ex:
            cs[N - 1] = np.nan
    return [est_a_i1, est_sigma_i1, j, time.time() - timing]

def ins_w_2_obs_bb(cs, est_a, est_sigma, level_c):
    timing = time.time()
    est_a_i = est_a
    est_sigma_i = est_sigma
    est_a_i1 = 1
```

```
est_sigma_i1 = 2
iter = 0
last_true_elem = 0
p = 0
s = 0
first_elems_ex = [True, True]

while math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                            est_sigma_i) >= eps:
    est_sample = []
    if iter != 0:
        est_a_i = est_a_i1
        est_sigma_i = est_sigma_i1
    for i in range(2):
        if cs[i] is np.nan:
            first_elems_ex[i] = False
            temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
            while temp < level_c:
                temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
            cs[i] = temp
    for i in range(N):
        if cs[i] is not np.nan:
            last_true_elem = cs[i]
            s = 0
            est_sample.append(last_true_elem)
        else:
            s += 1
            x_t_p = last_true_elem
            r = s + 1
            while cs[i - r] is np.nan:
                r += 1
            p = r - s
            mu_i = ((est_a_i ** (r - p)) - est_a_i ** (r + p)) * x_t_p /
            (1 - est_a_i ** (2 * p))
            sigma_i = est_sigma_i * (1 - (a ** (2 * r - 2 * p) - a **
             (2 * r))) /(1 - est_a_i ** (2 * p))
            normal_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
             (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
            num_of_it = 1
            while normal_var < level_c and num_of_it <= 1000:
                num_of_it += 1
                normal_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                 (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
            if num_of_it > 100000:
                exit()
```

```
                num_of_it = 0
                est_sample.append(normal_var)
        est_a_i1, est_sigma_i1 = est_coef_YW(est_sample)
        iter += 1
        for i in range(2):
            if not first_elems_ex[i]:
                cs[i] = np.nan
    return [est_a_i1, est_sigma_i1, iter, time.time() - timing]


def ins_w_2_obs_bb_new(cs, est_a, est_sigma, level_c):
    timing = time.time()
    est_a_i = est_a
    est_sigma_i = est_sigma
    est_a_i1 = 1
    est_sigma_i1 = 2
    iter = 0
    last_true_elem = 0
    p = 0
    s = 0
    first_elems_ex = [True, True]

    while math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                            est_sigma_i) >= eps:
        est_sample = []
        if iter != 0:
            est_a_i = est_a_i1
            est_sigma_i = est_sigma_i1
        for i in range(2):
            if cs[i] is np.nan:
                first_elems_ex[i] = False
                temp = random.normalvariate(0, (est_sigma_i /
                                        (1 - est_a_i ** 2)) ** 0.5)
                while temp < level_c:
                    temp = random.normalvariate(0, (est_sigma_i /
                                        (1 - est_a_i ** 2)) ** 0.5)
                cs[i] = temp
        for i in range(N):
            if cs[i] is not np.nan:
                last_true_elem = cs[i]
                est_sample.append(last_true_elem)
            else:
                x_t_p = last_true_elem
                r = 2
                p = 1
                mu_i = ((est_a_i ** (r - p)) - est_a_i ** (r + p)) * x_t_p /
                                        (1 - est_a_i ** (2 * p))
```

```
                sigma_i = est_sigma_i * (1 - (a ** (2 * r - 2 * p) -
                    a ** (2 * r))) / (1 - est_a_i ** (2 * p))
                normal_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                                (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                num_of_it = 1
                while normal_var < level_c and num_of_it <= 1000:
                    num_of_it += 1
                    normal_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                              (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                if num_of_it > 100000:
                    exit()
                num_of_it = 0
                est_sample.append(normal_var)
        est_a_i1, est_sigma_i1 = est_coef_YW(est_sample)
        iter += 1
        for i in range(2):
            if not first_elems_ex[i]:
                cs[i] = np.nan
    return [est_a_i1, est_sigma_i1, iter, time.time() - timing]


def ins_w_2_obs_ff(cs, est_a, est_sigma, level_c):
    timing = time.time()
    est_a_i = est_a
    est_sigma_i = est_sigma
    est_a_i1 = 1
    est_sigma_i1 = 2
    iter = 0
    last_elems_ex = [True, True]

    while math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                                    est_sigma_i) >= eps:
        est_sample = []
        if iter != 0:
            est_a_i = est_a_i1
            est_sigma_i = est_sigma_i1
        for i in range(2, 0, -1):
            if cs[N - i] is np.nan:
                last_elems_ex[2 - i] = False
                temp = random.normalvariate(0, (est_sigma_i /
                                        (1 - est_a_i ** 2)) ** 0.5)
                while temp < level_c:
                    temp = random.normalvariate(0, (est_sigma_i /
                                        (1 - est_a_i ** 2)) ** 0.5)
                cs[N - i] = temp
        for i in range(N):
            if cs[i] is not np.nan:
```

```
                est_sample.append(cs[i])
            else:
                q = 1
                while cs[i + q] is np.nan:
                    q += 1
                x_t_q = cs[i + q]
                s = q + 1
                while cs[i + s] is np.nan:
                    s += 1
                mu_i = ((est_a_i ** q) - est_a_i ** (2 * s - q)) * x_t_q /
                 (1 - est_a_i ** (2 * (s - q)))
                sigma_i = est_sigma_i * (1 - (est_a_i ** (2 * q) -
                 est_a_i ** (2 * s)) / (1 - est_a_i ** (2 * (s - q))))
                normal_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                num_of_it = 1
                while normal_var < level_c and num_of_it <= 1000:
                    num_of_it += 1
                    normal_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                     (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                if num_of_it > 100000:
                    exit()
                num_of_it = 0
                est_sample.append(normal_var)
        est_a_i1, est_sigma_i1 = est_coef_YW(est_sample)
        iter += 1
        for i in range(2, 0, -1):
            if not last_elems_ex[2 - i]:
                cs[N - i] = np.nan
    return [est_a_i1, est_sigma_i1, iter, time.time() - timing]


def ins_w_3_obs(cs, est_a, est_sigma, level_c):
    timing = time.time()
    est_a_i = est_a
    est_sigma_i = est_sigma
    est_a_i1 = 1
    est_sigma_i1 = 2
    iteration = 0
    m = 0
    h = 0
    q = 0
    first_elems_ex = [True, True]
    last_elem_ex = True

    while math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                                    est_sigma_i) >= eps:
```

```
est_sample = []
if iteration != 0:
    est_a_i = est_a_i1
    est_sigma_i = est_sigma_i1

for i in range(2):
    if cs[i] is np.nan:
        first_elems_ex[i] = False
        temp = random.normalvariate(0, (est_sigma_i /
                                 (1 - est_a_i ** 2)) ** 0.5)
        while temp < level_c:
            temp = random.normalvariate(0, (est_sigma_i /
                                 (1 - est_a_i ** 2)) ** 0.5)
        cs[i] = temp

if cs[N - 1] is np.nan:
    last_elem_ex = False
    temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
    while temp < level_c:
        temp = random.normalvariate(0, (est_sigma_i /
                                (1 - est_a_i ** 2)) ** 0.5)
    cs[N - 1] = temp

for index in range(N):
    if cs[index] is not np.nan:
        last_te = cs[index]
        s = 0
        q = 1
        est_sample.append(last_te)
    else:
        s += 1
        x_t_q = 0
        m = s + 1
        while cs[index - m] is np.nan:
            m += 1
        h = m - s
        x_t_h = last_te
        x_t = est_sample[index - m]
        if s == 1:
            while cs[index + q] is np.nan:
                q += 1
            x_t_q = cs[index + q]
            q += m
        mu_i = ((est_a_i ** (2 * q + h - m) - est_a_i ** (2 * q -
         h - m) + est_a_i ** (m - h) - est_a_i ** (m + h)) * x_t_h +
         ((est_a_i ** (q + m) + est_a_i ** (q - m) -
```

```
                    est_a_i ** (q + m - 2 * h) - est_a_i ** (q +
                      2 * h - m))) * x_t_q) / (1 + est_a_i ** (2 * q) -
                         est_a_i ** (2 * h) - est_a_i ** (2 * q - 2 * h))
                    sigma_i = est_sigma_i * (1 + 3 * est_a_i ** (2 * q) -
                     est_a_i ** (2 * m) - est_a_i ** (2 * h) - 3 *
                      est_a_i ** (2 * q - 2 * h) + est_a_i ** (2 * m -
                       2 * h) + est_a_i ** (2 * q - 2 * m) - est_a_i **
                       (2 * q + 2 * h - 2 * m)) / (1 + est_a_i ** (2 * q)
                        - est_a_i ** (2 * h) - est_a_i ** (2 * q - 2 * h))
                    norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                    (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                    noi = 1
                    while norm_var < level_c and noi < 10000:
                        noi += 1
                        norm_var = random.normalvariate(mu_i / (1 - est_a_i ** 2),
                        (sigma_i / (1 - est_a_i ** 2)) ** 0.5)
                    noi = 0
                    est_sample.append(norm_var)
            est_a_i1, est_sigma_i1 = est_coef_YW(est_sample)
            iteration += 1
            for i in range(2):
                if not first_elems_ex[i]:
                    cs[i] = np.nan
            if not last_elem_ex:
                cs[N - 1] = np.nan
        return [est_a_i1, est_sigma_i1, iteration, time.time() - timing]

    def est_coef_YW(sample):
        c_0 = c_1 = 0
        sample_len = len(sample)
        for j in range(0, sample_len):
            c_0 += sample[j] ** 2
            if j != len(sample) - 1:
                c_1 += sample[j] * sample[j + 1]
        c_0 /= sample_len
        c_1 /= (sample_len - 1)
        est_a = c_1 / c_0
        est_sigma = c_0 - est_a * c_1
        return est_a, est_sigma
```

ПРИЛОЖЕНИЕ Б. Листинг программы метода вставки

```
def count_reverse_covariance_submatr(indexes, dim, est_a, est_sigma):
    rev_cov = np.zeros((dim, dim))
    factor = (1 - est_a ** 2) / est_sigma ** 2
    dim_a = dim - 1
    coef_a_i = count_a_i([index + 1 for index in indexes], dim_a, est_a)
    rev_cov[0][0] = 1 / (1 - coef_a_i[0] ** 2)
    rev_cov[0][1] = -coef_a_i[0] / (1 - coef_a_i[0] ** 2)
    rev_cov[dim_a][dim_a] = 1 / (1 - coef_a_i[dim_a - 1] ** 2)
    rev_cov[dim_a][dim_a - 1] = -coef_a_i[dim_a - 1] / (1 -
                                        coef_a_i[dim_a - 1] ** 2)
    for i in range(1, dim_a):
        rev_cov[i][i - 1] = -coef_a_i[i - 1] / (1 - coef_a_i[i - 1] ** 2)
        rev_cov[i][i + 1] = -coef_a_i[i] / (1 - coef_a_i[i] ** 2)
        rev_cov[i][i] = (1 - coef_a_i[i - 1] ** 2 * coef_a_i[i] ** 2) / (
                (1 - coef_a_i[i - 1] ** 2) * (1 - coef_a_i[i] ** 2))
    return rev_cov * factor

def create_cov_submatr(obs_ind, pass_ind, dim, est_a, est_sigma):
    big_cov = count_covariance(dim, est_a, est_sigma)
    obs_dim = len(obs_ind)
    pass_dim = len(pass_ind)
    matr_pass_x = ([big_cov[i][:] for i in pass_ind])
    matr_obs_x = ([big_cov[i][:] for i in obs_ind])
    Sigma_oo = np.array([[matr_obs_x[i][j] for j in obs_ind] for i in
                                                        range(obs_dim)])
    Sigma_cc = np.array([[matr_pass_x[i][j] for j in pass_ind] for i in
                                                      range(pass_dim)])
    Sigma_co = np.array([[matr_pass_x[i][j] for j in obs_ind] for i in
                                                        range(pass_dim)])
    Sigma_oc = np.array([[matr_obs_x[i][j] for j in pass_ind] for i in
                                                        range(obs_dim)])

    return Sigma_oo, Sigma_cc, Sigma_co, Sigma_oc


def generate_Y(threshold_vect):
    Y = []
    for level in threshold_vect:
        rand_value = np.random.normal()
        while (rand_value <= level):
            rand_value = np.random.normal()
        Y.append(rand_value)
    return np.array(Y)
```

```
def ins_all_obs(cs, est_a, est_sigma, level_c):
    vect_level_c = np.array([level_c for i in
                range(len(np.where(np.isnan(cs))[0]))]).transpose()
    timing = time.time()
    est_a_i = est_a
    est_sigma_i = est_sigma
    est_a_i1 = 1
    est_sigma_i1 = 2
    k = 0
    cens_indexes = np.where(np.isnan(np.array(cs)))[0]
    obs_indexes = np.where(~np.isnan(np.array(cs)))[0]
    x_c_row = [x for x in cs if x is not np.nan]
    x_c_column = np.array(x_c_row).transpose()
    while math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                                    est_sigma_i) >= eps:
        if math.fabs(est_a_i1 - est_a_i) + math.fabs(est_sigma_i1 -
                                                    est_sigma_i) > 5:
            return [np.nan, np.nan, np.nan, np.nan]
        est_sample = cs.copy()
        if k != 0:
            est_a_i = est_a_i1
            est_sigma_i = est_sigma_i1
        Sigma_oo, Sigma_cc, Sigma_co, Sigma_oc = create_cov_submatr(obs_indexes,
                cens_indexes, N, est_a_i, est_sigma_i)
        inv_Sigma_oo = count_reverse_covariance_submatr(obs_indexes,
                    len(obs_indexes), est_a_i, est_sigma_i)
        Sigma_co_inv_oo_mult = Sigma_co.dot(inv_Sigma_oo)
        Mu_cc_o = Sigma_co_inv_oo_mult.dot(x_c_column)
        Sigma_cc_o = Sigma_cc - Sigma_co_inv_oo_mult.dot(Sigma_oc)
        cholesky_matr = linalg.cholesky(Sigma_cc_o)
        threshold_vect = linalg.inv(cholesky_matr).dot(vect_level_c - Mu_cc_o)
        Y = generate_Y(threshold_vect)
        X = cholesky_matr.dot(Y) + Mu_cc_o
        for index in range(len(X)):
            est_sample[cens_indexes[index]] = X[index]
        est_a_i1, est_sigma_i1 = est_coef_YW(est_sample)
        k += 1

    return [est_a_i1, est_sigma_i1, k, time.time() - timing]
```

ПРИЛОЖЕНИЕ В. Листинг программы приближённого метода максимального правдоподобия

```
def f_x_i_for_denom(x_i, inverse_sigma_matr):
    try:
        x_i_T = x_i.transpose()
        precommon_part_of_three_integrals = x_i_T.dot(inverse_sigma_matr)
        common_part_of_three_integrals = math.exp(
            -0.5 * precommon_part_of_three_integrals.dot(x_i))

    except OverflowError:
        print(level_cs, x_i, inverse_sigma_matr,
         precommon_part_of_three_integrals, -0.5 *
          precommon_part_of_three_integrals.dot(x_i))

    return common_part_of_three_integrals


def double_integral_midpoint_rule_denom(inverse_sigma_matr, a):
    int_top_value = a + 5
    dx = (int_top_value - a) / num_of_splitting

    integral_sum_f3 = 0.0

    for i in range(num_of_splitting):
        for j in range(num_of_splitting):
            x_midpoint = a + dx * (i + 0.5)
            x_i = np.array([[x_midpoint], [x_midpoint]])

            f3_value = f_x_i_for_denom(x_i, inverse_sigma_matr)

            integral_sum_f3 += f3_value

    integral_sum_f3 *= dx ** 2

    return integral_sum_f3


def composite_trapezoidal_rule_denom(x_i_org, inverse_sigma_matr, a):
    int_top_value = a + 5
    h = (int_top_value - a) / num_of_splitting
    x_0 = np.where(np.isnan(x_i_org), a, x_i_org)
    x_n = np.where(np.isnan(x_i_org), int_top_value, x_i_org)
```

```
    integrative_function_for_x_0_third = f_x_i_for_denom(x_0, inverse_sigma_matr)
    integrative_function_for_x_n_third = f_x_i_for_denom(x_n, inverse_sigma_matr)

    sum_of_internals_third = (integrative_function_for_x_0_third +
     integrative_function_for_x_n_third) / 2
    for i in range(1, num_of_splitting):
        x_i = np.where(np.isnan(x_i_org), a + i * h, x_i_org)
        temp_x_i_third = f_x_i_for_denom(x_i, inverse_sigma_matr)

        sum_of_internals_third += temp_x_i_third

    sum_of_internals_third *= h

    return sum_of_internals_third

def composite_trapezoidal_rule(x_i_org, inverse_sigma_matr,
 res_sigma_matr_first_eq, res_sigma_matr_sec_eq, a):
    int_top_value = a + 5
    h = (int_top_value - a) / num_of_splitting
    x_0 = np.where(np.isnan(x_i_org), a, x_i_org)
    x_n = np.where(np.isnan(x_i_org), int_top_value, x_i_org)

    integrative_function_for_x_0_first, integrative_function_for_x_0_sec,
     integrative_function_for_x_0_third = f_x_i_for_all_integrals(
        x_0, inverse_sigma_matr, res_sigma_matr_first_eq, res_sigma_matr_sec_eq)
    integrative_function_for_x_n_first, integrative_function_for_x_n_sec,
     integrative_function_for_x_n_third = f_x_i_for_all_integrals(
        x_n, inverse_sigma_matr, res_sigma_matr_first_eq, res_sigma_matr_sec_eq)

    sum_of_internals_first = (integrative_function_for_x_0_first +
     integrative_function_for_x_n_first) / 2
    sum_of_internals_second = (integrative_function_for_x_0_sec
+
 integrative_function_for_x_n_sec) / 2
    sum_of_internals_third = (integrative_function_for_x_0_third +
     integrative_function_for_x_n_third) / 2
    for i in range(1, num_of_splitting):
        x_i = np.where(np.isnan(x_i_org), a + i * h, x_i_org)
        temp_x_i_first, temp_x_i_sec, temp_x_i_third = f_x_i_for_all_integrals(
            x_i, inverse_sigma_matr, res_sigma_matr_first_eq,
             res_sigma_matr_sec_eq)

        sum_of_internals_first += temp_x_i_first
        sum_of_internals_second += temp_x_i_sec
        sum_of_internals_third += temp_x_i_third

    sum_of_internals_first *= h
```

```
    sum_of_internals_second *= h
    sum_of_internals_third *= h

    return sum_of_internals_first, sum_of_internals_second,
     sum_of_internals_third

def f_x_i_for_all_integrals(x_i, inverse_sigma_matr,
 res_sigma_matr_first_eq, res_sigma_matr_sec_eq):
    try:
        x_i_T = x_i.transpose()
        precommon_part_of_three_integrals = x_i_T.dot(inverse_sigma_matr)
        common_part_of_three_integrals = math.exp(
            -0.5 * precommon_part_of_three_integrals.dot(x_i))

        first_int_common_path_xit_big_sigma = x_i_T.dot(res_sigma_matr_first_eq)
        first_int_res = first_int_common_path_xit_big_sigma.dot(x_i)

        second_int_common_path_xit_big_sigmas = x_i_T.dot(res_sigma_matr_sec_eq)
        second_int_res = second_int_common_path_xit_big_sigmas.dot(x_i)
        first_int_res *= common_part_of_three_integrals
        second_int_res *= common_part_of_three_integrals
    except OverflowError:
        print(level_cs, x_i, inverse_sigma_matr,
         precommon_part_of_three_integrals,
              -0.5 * precommon_part_of_three_integrals.dot(x_i))

    return first_int_res, second_int_res, common_part_of_three_integrals‘


def double_integral_midpoint_rule(inverse_sigma_matr, res_sigma_matr_first_eq,
res_sigma_matr_sec_eq, a):
    int_top_value = a + 5
    dx = (int_top_value - a) / num_of_splitting

    integral_sum_f1 = 0.0
    integral_sum_f2 = 0.0
    integral_sum_f3 = 0.0

    for i in range(num_of_splitting):
        for j in range(num_of_splitting):
            x_midpoint = a + dx * (i + 0.5)
            x_i = np.array([[x_midpoint], [x_midpoint]])

            f1_value, f2_value, f3_value = f_x_i_for_all_integrals(x_i,
            inverse_sigma_matr, res_sigma_matr_first_eq, res_sigma_matr_sec_eq)

            integral_sum_f1 += f1_value
```

```
            integral_sum_f2 += f2_value
            integral_sum_f3 += f3_value

    integral_sum_f1 *= dx ** 2
    integral_sum_f2 *= dx ** 2
    integral_sum_f3 *= dx ** 2

    return integral_sum_f1, integral_sum_f2, integral_sum_f3


def count_first_approximation(cens_sample, j, k, t):
    (est_sigma_1_1_numerator, est_sigma_1_1_denominator, est_sigma_1_2_numerator,
     est_sigma_1_2_denominator, est_sigma_2_2_numerator,
      est_sigma_2_2_denominator) = 0, 0, 0, 0, 0, 0
    for i in range(t):
        x_1 = cens_sample[j + (i * k)]
        x_2 = cens_sample[j + (i * k) + 1]
        obs_x_1 = 0 if x_1 is np.nan else 1
        obs_x_2 = 0 if x_2 is np.nan else 1

        est_sigma_1_1_numerator += x_1 ** 2 if obs_x_1 == 1 else 0
        est_sigma_1_1_denominator += obs_x_1

        est_sigma_1_2_numerator += x_1 * x_2 if obs_x_1 * obs_x_2 == 1 else 0
        est_sigma_1_2_denominator += obs_x_1 * obs_x_2

        est_sigma_2_2_numerator += x_2 ** 2 if obs_x_2 == 1 else 0
        est_sigma_2_2_denominator += obs_x_2

    est_sigma_1_first = (est_sigma_1_1_numerator / est_sigma_1_1_denominator +
     est_sigma_2_2_numerator / est_sigma_2_2_denominator) / 2
    est_sigma_2_first = est_sigma_1_2_numerator / est_sigma_1_2_denominator

    return est_sigma_1_first, est_sigma_2_first

def logarithmic_maximum_likelihood_function(sigma_0, sigma_1, j, k, t,
cens_sample, level_c):
    determinant = sigma_0 ** 2 - sigma_1 ** 2
    sigma_matr = np.array([[sigma_0, sigma_1], [sigma_1, sigma_0]])
    inverse_sigma_matr = np.linalg.inv(sigma_matr)
    third_term, temp = 0, 0
    for i in range(t):
        index1 = j + (i * k)
        index2 = j + (i * k) + 1

        x_i = np.array([[cens_sample[index1]], [cens_sample[index2]]])
        if not np.isnan(x_i).any():
```

```
            temp = x_i.transpose().dot(inverse_sigma_matr)
            third_term -= temp.dot(x_i) / 2
        elif np.sum(np.isnan(x_i)) == 1:
            third_term += math.log(composite_trapezoidal_rule_denom(x_i,
             inverse_sigma_matr, level_c))
        else:
            third_term += math.log(double_integral_midpoint_rule_denom(
            inverse_sigma_matr, level_c))
    return -t * math.log(2 * math.pi) - t / 2 * math.log(determinant) +
    third_term

    def mlm_special_method(cs, temp_a, temp_sigma, level_c):
    timing = time.time()
    print("start interval method")
    k = int(math.sqrt(N))  # шаг между интервалами
    s = k  # количество вариантов деления выборки на интервалы
    t = int(N / k)  # количество интервалов в 1 варианте
    grid_step = 0.1  # при хождении по сетке

    sigma_0_est = []
    sigma_1_est = []
    gl_i = 0
    condition_to_stop = math.pow(2, -10)

    for j in range(k):
        alpha = 1
        first_eq_res, second_eq_res = 0, 0
        num_of_grad_iter = 0
        sigma_0, sigma_1 = count_first_approximation(cs, j, k, t)
        cur_value_of_max_likelihood_func =
         logarithmic_maximum_likelihood_function(sigma_0, sigma_1, j, k, t, cs,
          level_c)
        while (num_of_grad_iter < 20):
            num_of_grad_iter += 1
            diff_sq_sigmas = sigma_0 ** 2 - sigma_1 ** 2
            if diff_sq_sigmas > 0:
                sigma_matr = np.array([[sigma_0, sigma_1], [sigma_1, sigma_0]])
                inverse_sigma_matr = np.linalg.inv(sigma_matr)

                res_sigma_matr_first_eq = np.dot(inverse_sigma_matr,
                inverse_sigma_matr)
                temp_sec_eq_sigma = np.dot(inverse_sigma_matr,
                 np.array([[0, 1], [1, 0]]))
                res_sigma_matr_sec_eq = np.dot(temp_sec_eq_sigma,
                inverse_sigma_matr)
                try:
                    for i in range(t):
```

```
            gl_i = i
            index1 = j + (i * k)
            index2 = j + (i * k) + 1

            x_i = np.array([[cs[index1]], [cs[index2]]])

            if not np.isnan(cs[index1]) and not np.isnan(cs[index2]):
                res_row_and_matr_first = np.dot(x_i.transpose(),
                 res_sigma_matr_first_eq)
                res_row_and_matr_sec = np.dot(x_i.transpose(),
                 res_sigma_matr_sec_eq)
                first_eq_res += 0.5 * np.dot(res_row_and_matr_first,
                 x_i)
                second_eq_res += 0.5 * np.dot(res_row_and_matr_sec,
                 x_i)
            elif np.sum(np.isnan(x_i)) == 1:
                first_eq_first_int, second_eq_first_int, denom_int =
                 composite_trapezoidal_rule(x_i,
                  inverse_sigma_matr,
                  res_sigma_matr_first_eq,
                  res_sigma_matr_sec_eq,
                  level_c)
                first_eq_res += 0.5 * first_eq_first_int / denom_int
                second_eq_res += 0.5 * second_eq_first_int / denom_int
            else:
                first_eq_first_int, second_eq_first_int, denom_int =
                double_integral_midpoint_rule(
                    inverse_sigma_matr,
                    res_sigma_matr_first_eq,
                    res_sigma_matr_sec_eq,
                    level_c)
                first_eq_res += 0.5 * first_eq_first_int / denom_int
                second_eq_res += 0.5 * second_eq_first_int / denom_int

        except IndexError:
            print("ERROR", j, gl_i, k)
        first_eq_res -= t * sigma_0 / diff_sq_sigmas
        second_eq_res += t * sigma_1 / diff_sq_sigmas

        while alpha > condition_to_stop:
            sigma_0_k, sigma_1_k = sigma_0 + alpha * first_eq_res[0][0],
             sigma_1 + alpha * second_eq_res[0][0]
            if sigma_0_k ** 2 - sigma_1_k ** 2 > 0:
                new_value_of_max_likelihood_func =
                 logarithmic_maximum_likelihood_function(
                 sigma_0_k, sigma_1_k, j,k, t, cs, level_c)
```

```
                    if new_value_of_max_likelihood_func >
                     cur_value_of_max_likelihood_func:
                        alpha /= 2
                        sigma_0_k1, sigma_1_k1 = sigma_0 +
                         alpha * first_eq_res[0][0], sigma_1 + alpha * \
                                                  second_eq_res[0][0]
                        k1_new_value_of_likelihood_func =
                         logarithmic_maximum_likelihood_function(
                        sigma_0_k1,sigma_1_k1, j,k, t, cs, level_c)
                        if k1_new_value_of_likelihood_func >
                         new_value_of_max_likelihood_func:
                            cur_value_of_max_likelihood_func =
                            k1_new_value_of_likelihood_func
                            sigma_0 = sigma_0_k1
                            sigma_1 = sigma_1_k1
                            alpha = min(1, alpha * 4)
                            break
                        else:
                            cur_value_of_max_likelihood_func =
                             new_value_of_max_likelihood_func
                            sigma_0 = sigma_0_k
                            sigma_1 = sigma_1_k
                            alpha = min(1, alpha * 4)
                            break
                    else:
                        alpha /= 2
                else:
                    alpha /= 2

            first_eq_res, second_eq_res = 0, 0
            if alpha <= condition_to_stop:
                break
    sigma_0_est.append(sigma_0)
    sigma_1_est.append(sigma_1)

c_0 = sum(sigma_0_est) / len(sigma_0_est)
c_1 = sum(sigma_1_est) / len(sigma_1_est)

est_a, est_sigma = est_coef_YW_params(c_0, c_1)

print("end interval method")
return [est_a, est_sigma, 1, time.time() - timing]
```

ПРИЛОЖЕНИЕ Г. Листинг программы метода моментов

```
def moments_method_for_random_omissions(cs_sample, cs_a_ch, cs_sigma_ch,
level_cs):
    timing = time.time()
    print("start moments method for random omissions")
    cs_sample_array = np.array(cs_sample)

    O = (~np.isnan(cs_sample_array)).astype(int)
    mask = ~np.isnan(cs_sample_array)
    G_0 = np.sum((cs_sample_array[mask] ** 2) * O[mask]) / np.sum(O[mask])
    mask_1 = mask[:-1] & mask[1:]
    G_1 = np.sum(cs_sample_array[:-1][mask_1] * cs_sample_array[1:][mask_1] * O[:-1
    O[1:][mask_1]) / np.sum(O[:-1][mask_1] * O[1:][mask_1])

    est_a = G_1 / G_0
    est_sigma = G_0 - (G_1 ** 2) / G_0
    print("end moments method for random omissions")

    return [est_a, est_sigma, 1, time.time() - timing]
```