МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Самарский национальный исследовательский университет имени академика С.П.Королёва»

ИНСТИТУТ ИНФОРМАТИКИ, МАТМАТИКИ И ЭЛЕКТРОНИКИ

ФАКУЛЬТЕТ ИНФОРМАТИКИ

КАФЕДРА ТЕХНИЧЕСКОЙ КИБЕРНЕТИКИ

СТАТИСТИЧЕСКИЙ АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРОЦЕССОВ АВТОРЕГРЕССИИ И СКОЛЬЗЯЩЕГО СРЕДНЕГО

курсовая работа по дисциплине «**Теория случайных процессов**»

Вариант № 51

Выполнил:	Белоусов А.А.
Группа:	6309
№ зачетной книжки:	146166
Проверил:	Храмов А. Г.
Оценка:	_____
Дата:	_____

Самара 2018

РЕФЕРАТ

Курсовая работа по курсу «Теория случайных процессов» 57 страниц, 6 рисунков, 9 таблиц, 3 источника, 2 приложения.

МОМЕНТНЫЕ ФУНКЦИИ, МАТЕМАТИЧЕСКОЕ ОЖИДАНИЕ, ДИСПЕРСИЯ, КОРРЕЛЯЦИОННАЯ ФУНКЦИЯ, МОДЕЛЬ АВТОРЕГРЕСИИ, МОДЕЛЬ СКОЛЬЗЯЩЕГО СРЕДНЕГО, МОДЕЛЬ ARMA

Объектом исследований является выборка из $n = 5000$ последовательных значений стационарного в широком смысле эргодичного случайного процесса с дискретным временем (временной ряд).

Цель работы – оценка моментных функций, построение и исследование моделей авторегрессии, скользящего среднего, смешанных моделей, моделирование случайных процессов.

Были расчитаны выборочные моментные функции, построены и исследованы модели АР, СС, АРСС. Проведено моделирование процессов АР(3), СС(1), АРСС(3, 1).

СОДЕРЖАНИЕ

1 ЗАДАНИЕ

1.1 Оценивание моментных функций

Изобразить графически фрагмент исходного случайного процесса (СП). Оценить моментные функции (МФ) исходного случайного процесса, рассчитав выборочные среднее, дисперсию и нормированную корреляционную функцию (НКФ). Оценить интервал корреляции СП. Изобразить графически оценку НКФ исходного СП.

1.2 Построение и исследование моделей авторегрессии

Построить модели авторегрессии АР(M) = АРСС(M, 0) порядков M = 0, 1, 2, 3 (всего 4 модели) на основе решения системы уравнений Юла–Уокера. Для каждой модели рассчитать теоретические НКФ выходной последовательности. На основе сравнения выборочной НКФ и теоретических НКФ выбрать лучшую модель СП в классе моделей АР.

1.3 Построение и исследование моделей скользящего среднего

Построить модели скользящего среднего СС(N) = АРСС(0, N) порядков N = 0, 1, 2, 3 (всего 4 модели) на основе решения системы нелинейных уравнений. Для каждой модели рассчитать теоретические НКФ выходной последовательности. На основе сравнения выборочной НКФ и теоретических НКФ выбрать лучшую модель СП в классе моделей СС.

1.4 Построение и исследование смешанных моделей авторегрессии – скользящего среднего

Построить смешанные модели авторегрессии – скользящего среднего АРСС(M, N) до третьего порядка включительно (M = 1, 2, 3; N = 1, 2, 3) (всего 9 моделей) одним из методов, описанным в приложении А.3. Рассчитать теоретические НКФ выходной последовательности для каждой модели АРСС. На основе сравнения исходной выборочной и теоретических НКФ выбрать лучшую модель СП в классе смешанных моделей АРСС. Исследовать на устойчивость смешанные модели.

1.5 Сравнительный анализ построенных моделей

Для каждой из трёх лучших моделей (АР, СС, АРСС) записать системы уравнений для расчёта параметров модели, записать системы уравнений для расчёта теоретической КФ, смоделировать СП, рассчитать выборочные МФ, сравнить их с выборочными МФ исходного СП и с теоретическими МФ. Для каждой из этих трёх моделей сравнить графически НКФ: (1) выборочную исходного СП, (2) теоретическую, (3) выборочную смоделированного СП.

1.6 Итоговая таблица сравнения моделей АРСС

Изготовить таблицу сравнения МФ и расчёта качества для трёх лучших моделей. Изобразить графически фрагмент реализации СП, сгенерированного по наилучшей модели.

2 ИСХОДНЫЕ ДАННЫЕ

Дана реализация стационарного в широком смысле эргодического случайного процесса с дискретным временем (стационарная случайная последовательность, временной ряд) – выборка из $n = 5000$ последовательных значений (отсчётов) процесса:

-26.364 -31.116 -16.294 -33.269 -12.440 -36.868 -21.972 -29.761 -14.903 -33.905 -13.594 -37.352 -20.000 -32.988 -13.362 -23.077 -14.026 -23.744 -31.442 -39.546 -36.071 -20.985 -21.972 -23.090 -25.598 -20.838 -21.724 -21.249 -27.790 -23.872 -27.504 -27.615 -29.436 -31.517 -17.928

...

-29.333 -23.756 -16.822 -27.577 -20.445 -17.546 -30.783 -18.043 -33.864 -18.692 -35.802 -15.114 -36.468 -17.033 -41.076 -14.229 -36.877 -18.515 -36.372 -10.348 -28.443 -13.548 -38.428 -21.716 -37.438 -13.258 -33.314 -7.763 -33.864 -15.854 -35.060 -24.056 -29.442 -21.433 -25.555

3 ОЦЕНИВАНИЕ МОМЕНТНЫХ ФУНКЦИЙ

3.1 Графическое представление

Графически изобразим фрагмент исходного случайного процесса на рисунке 1.

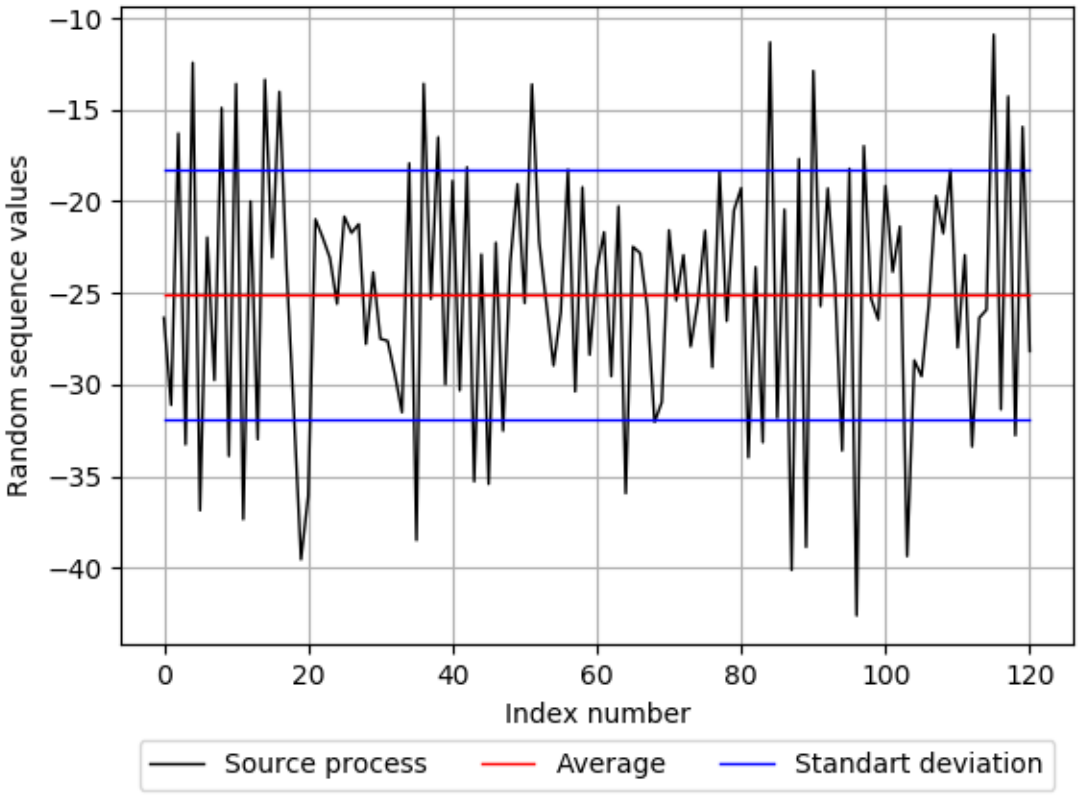

Рисунок 1 – График исходного случайного процесса

3.2 Моментные функции

Пусть дана выборка $X_1 ... X_n$ из n отсчётов стационарного в широком смысле эргодического дискретного случайного процесса. Оценим моментные функции этого процесса, рассчитав:

1. Выборочное среднее
2. Выборочную дисперсию
3. Нормированную корреляционную функцию (НКФ).

1. Выборочное среднее $\overline{X}$ - это оценка математического ожидания $\mathbb{M}(X)$. Формула для

расчётов:

$$\overline{X} = \frac{1}{n}\sum_{k=1}^{n} X_k, \tag{1}$$

где X_k - элементы выборки при $k \in [0, n]$, а n - размер выборки.

Используя программу из приложения А, получаем $\overline{X} = -25.072198200000003$.

На рисунке 1 показано значение $\overline{X}$.

2. Выборочная дисперсия S^2 - это оценка дисперсии $D(\overline{X}) = D_x$ распределения на основе выборки. Различают два вида дисперсии – выборочную и несмещённую.

   Формула для расчёта выборочной дисперсии:

$$S^2{}_n = \frac{1}{n}\sum_{k=1}^{n} (X_k - \overline{X})^2. \tag{2}$$

   Формула для расчёта несмещённой или исправленной дисперсии:

$$\hat{S}_n^2 = \frac{1}{n-1}\sum_{k=1}^{n} (X_k - \overline{X})^2. \tag{3}$$

   В данном случае была использована исправленная дисперсия, т.к. она даёт несмещённую оценку дисперсии:

$$M(\hat{S}_n^2) = D_x. \tag{4}$$

   Применяя формулу (3), получаем $\hat{S}_n^2 = 45.94676292851676$.

   Для вычисления среднеквадратического отклонения (СКО) используется следующая формула:

$$\hat{S}_n = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n} (X_k - \overline{X})^2}. \tag{5}$$

   Принимая во внимание значение $\hat{S}_n^2$, получаем $\hat{S}_n = 6.778404157950215$.

   На рисунке 1 отображены значения $\overline{X} + \hat{S}_n$ и $\overline{X} - \hat{S}_n$.

3. Формула для вычисления оценки исправленой выборочной корреляционной функции $\hat{R}_\xi(k)$:

$$\hat{R}_\xi(k) = \frac{1}{n-k-1}\sum_{j=1}^{n-k} (X_j - \overline{X})(X_{j+k} - \overline{X}). \tag{6}$$

В таблице 1 представлены значения этой функции для $k = 0, 1, \dots 10$.

Оценка нормированной корреляционной функции $\hat{r}_\xi(k)$ определяется следующим образом:

$$\hat{r}_\xi(k) = \frac{\hat{R}_\xi(k)}{\hat{R}_\xi(0)} = \frac{\hat{R}_\xi(k)}{\hat{S}_n^2}. \tag{7}$$

Её значения для $k = 0, 1, \dots 10$ также представлены в таблице 1.

Таблица 1 – Значения выборочных КФ и НКФ

k	$\hat{R}_\xi(k)$	$\hat{r}_\xi(k)$
0	45.94676	1.0
1	-16.83377	-0.36638
2	20.06587	0.43672
3	-20.94726	-0.4559
4	11.22639	0.24433
5	-14.17409	-0.30849
6	8.02627	0.17469
7	-8.94403	-0.19466
8	5.63892	0.12273
9	-5.55074	-0.12081
10	3.49849	0.07614

Оценим интервал корреляции:

$$\tau_к = min\{\tau : \forall(m > \tau)\, |r_\eta(m)| < \frac{1}{e}\}, \tag{8}$$

полагая что $m << n$.

Используя программу из приложения А, получаем значение $\tau_к = 3$. На рисунке 2 изображена оценка интервала корреляции исходного случайного процесса.

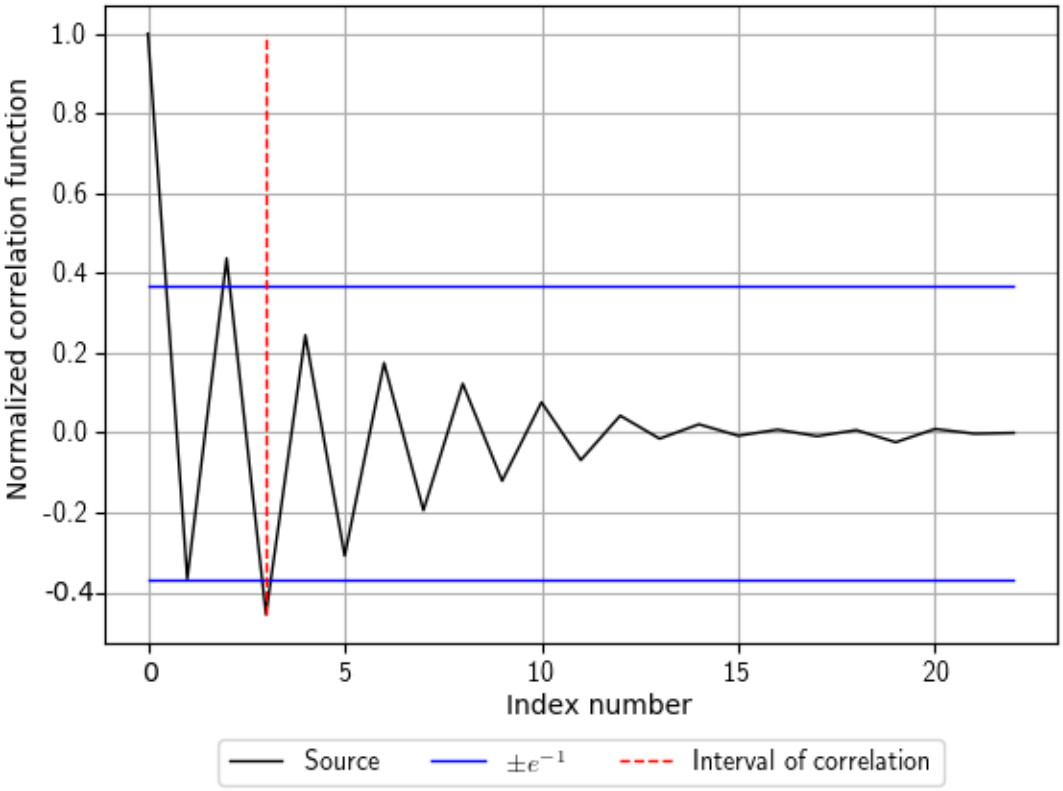

Рисунок 2 – Графическая оценка интервала корреляции

4 ПОСТРОЕНИЕ И ИССЛЕДОВАНИЕ МОДЕЛЕЙ АВТОРЕГРЕССИИ

4.1 Модель АРСС(M, 0)

Модель авторегрессии порядка M в общем виде может быть записанна как:

$$\eta_n = \beta_1\eta_{n-1} + \beta_2\eta_{n-2} + \cdots + \beta_M\eta_{n-M} + \alpha_0\xi_n, \tag{9}$$

где ξ – входная некоррелированная случайная последовательность с нулевым математическим ожиданием и единичной дисперсией ($\mathbb{M}(\xi_n) = 0$, $\mathbb{M}(\xi_n^2) = 1$, $\mathbb{M}(\xi_k\xi_n) = \delta_{kn}$), η – выходная случайная, стационарная в широком смысле последовательность с корреляционной функцией $R_\eta(m)$ и нулевым математическим ожиданием, $\{M, \alpha_0, \beta_1, \beta_2, \ldots, \beta_M\}$ – параметры модели авторегрессии.

Построим модели авторегрессии для $M = 0, 1, 2, 3$ на основе решения системы уравнений Юла-Уокреа. Для решения линейных систем был использован метод numpy.linalg.solve из библиотеки numpy. Выпишем системы уравнений.

Для модели АРСС(0, 0) получаем:

$$R_\eta(0) = \alpha_0^2 \tag{10}$$

Решая уравнение (10), получаем $\alpha_0 = 6.77840416$.

Для модели АРСС(1, 0) получаем:

$$\begin{cases} R_\eta(0) = \beta_1 R_\eta(1) + \alpha_0^2 \\ R_\eta(1) = \beta_1 R_\eta(0) \end{cases} \tag{11}$$

Решая систему (11), получаем $\alpha_0 = 6.30708201$, $\beta_1 = -0.36637548$.

Для модели АРСС(2, 0) получаем:

$$\begin{cases} R_\eta(0) = \beta_1 R_\eta(1) + \beta_2 R_\eta(2) + \alpha_0^2 \\ R_\eta(1) = \beta_1 R_\eta(0) + \beta_2 R_\eta(1) \\ R_\eta(2) = \beta_1 R_\eta(1) + \beta_2 R_\eta(0) \end{cases} \tag{12}$$

Решая систему (12), получаем $\alpha_0 = 5.90959855$, $\beta_1 = -0.23836838$, $\beta_2 = 0.34938774$.

Для модели АРСС(3, 0) получаем:

$$\begin{cases} R_\eta(0) = \beta_1 R_\eta(1) + \beta_2 R_\eta(2) + \beta_3 R_\eta(3) + \alpha_0^2 \\ R_\eta(1) = \beta_1 R_\eta(0) + \beta_2 R_\eta(1) + \beta_3 R_\eta(2) \\ R_\eta(2) = \beta_1 R_\eta(1) + \beta_2 R_\eta(0) + \beta_3 R_\eta(1) \\ R_\eta(3) = \beta_1 R_\eta(2) + \beta_2 R_\eta(1) + \beta_3 R_\eta(0) \end{cases} \tag{13}$$

Решая систему (13), получаем $\alpha_0 = 5.64763391$, $\beta_1 = -0.1354962$, $\beta_2 = 0.27920361$, $\beta_3 = -0.29443558$.

Проверим полученные модели на устойчивость. Для этого выясним лежат ли все корни z характерестического уравнения (14) внутри единичной окружности $|z| < 1$ на комплексной плоскости:

$$\sum_{k=1}^{M} \beta_k z^{-k} = 1. \tag{14}$$

Рассмотрим каждую модель:

- Модель APCC(0, 0) устойчива всегда.
- Модель APCC(1, 0) устойчива тогда и только тогда, когда $|\beta_1| < 1$:

  $$|-0.36637548| < 1 \Rightarrow \text{Модель устойчива.}$$

- Модель APCC(2, 0) устойчива тогда и только тогда, когда $|\beta_2| < 1$, $|\beta_1| < 1 - \beta_2$:

  $$\left.\begin{array}{l} |0.34938774| < 1 \\ |-0.23836838| < 0.65061226 \end{array}\right\} \Rightarrow \text{Модель устойчива.}$$

- Модель APCC(3, 0) устойчива тогда и только тогда, когда $|\beta_3| < 1$, $|\beta_1 + \beta_3| < 1 - \beta_2$, $|\beta_2 + \beta_1\beta_3| < |1 - \beta_3^2|$:

  $$\left.\begin{array}{l} |-0.29443558| < 1 \\ |-0.42993178| < 0.72079639 \\ |0.31909851| < 0.91330769 \end{array}\right\} \Rightarrow \text{Модель устойчива.}$$

4.2 Расчёт теоретической НКФ выходной последовательности

Для каждой модели рассчитаем теоретическую НКФ выходной последовательности. Для этого воспользуемся следующей формулой:

$$r_\eta^{\mathrm{T}}(M + k) = \sum_{j=1}^{M} b_j r_\eta^{\mathrm{T}}(M + k - j), k \geq 1, \tag{15}$$

учитывая что первые $M + 1$ значений теоретической НКФ нам уже известны, т.к. они совпадают с первыми значениями выборочной НКФ для исходных данных.

Полученные значения, для $k = 0, 1, 2, \ldots, 10$, представлены в таблице 2.

Таблица 2 – Значения теоретических НКФ для моделей АР(M)

$r_\eta^{\mathrm{T}}(k)$	АР(0)	АР(1)	АР(2)	АР(3)
$r_\eta^{\mathrm{T}}(0)$	1.00000	1.00000	1.00000	1.00000
$r_\eta^{\mathrm{T}}(1)$	0.00000	-0.36638	-0.36638	-0.36638
$r_\eta^{\mathrm{T}}(2)$	0.00000	0.13423	0.43672	0.43672
$r_\eta^{\mathrm{T}}(3)$	0.00000	-0.04918	-0.23211	-0.4559
$r_\eta^{\mathrm{T}}(4)$	0.00000	0.01802	0.20791	0.29158
$r_\eta^{\mathrm{T}}(5)$	0.00000	-0.0066	-0.13066	-0.29538
$r_\eta^{\mathrm{T}}(6)$	0.00000	0.00242	0.10379	0.25567
$r_\eta^{\mathrm{T}}(7)$	0.00000	-0.00089	-0.07039	-0.20297
$r_\eta^{\mathrm{T}}(8)$	0.00000	0.00032	0.05304	0.18586
$r_\eta^{\mathrm{T}}(9)$	0.00000	-0.00012	-0.03724	-0.15713
$r_\eta^{\mathrm{T}}(10)$	0.00000	0.00004	0.02741	0.13294

4.3 Выбор налучшей модели случайного процесса в классе моделей АР

Выбор лучшей модели проводится на основе анализа нормированных корреляционных функций с использованием критерия среднего квадратичного отклонения по первым десяти отсчётам:

$$\epsilon^2 = \sum_{k=1}^{10} (r^{\mathrm{T}}(k) - \hat{r}(k))^2, \tag{16}$$

где $\hat{r}(k)$ – выборочная НКФ исходного процесса, r^{T} – теоретическая НКФ. Результаты представлены в таблице 3. По результатам делаем следующий вывод: среди данных моделей, процесс лучше всего описывает модель АР(3).

Таблица 3 – Результаты построения моделей АР(M)

Порядок модели	Параметры модели				Погрешность модели
M	β_1	β_2	β_3	α_0	ϵ^2
0	-	-	-	6.77840	0.79153
1	-0.36637	-	-	6.30708	0.50184
2	-0.23837	0.34939	-	5.90960	0.11772
3	-0.13549	0.27920	-0.29443	5.64763	0.01756

5 ПОСТРОЕНИЕ И ИССЛЕДОВАНИЕ МОДЕЛЕЙ СКОЛЬЗЯЩЕГО СРЕДНЕГО

5.1 Модель APCC(0, N)

Модель скользящего среднего порядка N в общем виде может быть записана как:

$$\eta_n = \alpha_0\xi_n + \alpha_1\xi_{n-1} + \cdots + \alpha_N\xi_{n-N}, \tag{17}$$

где ξ – входная некоррелированная случайная последовательность с нулевым математическим ожиданием и единичной дисперсией ($\mathbb{M}(\xi_n) = 0$, $\mathbb{M}(\xi_n^2) = 1$, $\mathbb{M}(\xi_k\xi_n) = \delta_{kn}$), η – выходная случайная, стационарная в широком смысле последовательность с корреляционной функцией $R_\eta(m)$ и нулевым математическим ожиданием, $\{N, \alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_N\}$ – параметры модели скользящего среднего.

Построим модели скользящего среднего для $N = 0, 1, 2, 3$ на основе решения системы нелинейных уравнений. Решение будем получать исходя из известных значений выборочной корреляционной функции $R_\eta(m)$ и следующей системы нелинейных уравнений:

$$R_\eta(m) = \begin{cases} \sum\limits_{n=1}^{N-m} \alpha_n\alpha_{n+M} & \text{если } m \le N \\ 0 & \text{иначе} \end{cases} \tag{18}$$

Выпишем системы для CC(0), CC(1), CC(2) и CC(3). Решим эти системы используя метод простых итераций из приложения Б.

- Для CC(0) получаем:

$$R_\eta(0) = \alpha_0^2 \to \alpha_0^2 = 45.94676 \to \alpha_0 = 6.77840.$$

- Для CC(1) получаем:

$$\begin{cases} R_\eta(0) = \alpha_0^2 + \alpha_1^2 \\ R_\eta(1) = \alpha_0\alpha_1 \end{cases} \to \alpha_0 = -2.70925, \alpha_1 = 6.21343.$$

- Для CC(2) получаем:

$$\begin{cases} R_\eta(0) = \alpha_0^2 + \alpha_1^2 + \alpha_2^2 \\ R_\eta(1) = \alpha_0\alpha_1 + \alpha_1\alpha_2 \\ R_\eta(2) = \alpha_0\alpha_2 \end{cases} \to \alpha_0 = 5.317996, \alpha_1 = -1.85165, \alpha_2 = 3.77320.$$

- Для CC(3) получаем:

$$\begin{cases} R_\eta(0) = \alpha_0^2 + \alpha_1^2 + \alpha_2^2 + \alpha_3^2 \\ R_\eta(1) = \alpha_0\alpha_1 + \alpha_1\alpha_2 + \alpha_2\alpha_3 \\ R_\eta(2) = \alpha_0\alpha_2 + \alpha_1\alpha_3 \\ R_\eta(3) = \alpha_0\alpha_3 \end{cases} \rightarrow \text{Система не имеет решения.}$$

При решении систем с помощью программы из приложения А было установлено: модели CC(3) в данном случае не существует.

5.2 Расчёт теоретической НКФ выходной последовательности

Для каждой модели рассчитаем теоретическую НКФ выходной последовательности. Для этого воспользуемся следющим фактом: первые $N+1$ значений теоретической НКФ сопадают со значениями выборочной НКФ исходного процесса, а остальные равны нулю (это следует из некоррелированности входного процесса ξ).

Значения, для $k = 0, 1, 2, \ldots, 10$, представлены в таблице 4.

Таблица 4 – Значения теоретических НКФ для моделей CC(N)

$r_\eta^{\mathrm{T}}(k)$	CC(0)	CC(1)	CC(2)
$r_\eta^{\mathrm{T}}(0)$	1.00000	1.00000	1.00000
$r_\eta^{\mathrm{T}}(1)$	0.00000	-0.366375	-0.366375
$r_\eta^{\mathrm{T}}(2)$	0.00000	0.00000	0.43672
$r_\eta^{\mathrm{T}}(3)$	0.00000	0.00000	0.00000
$r_\eta^{\mathrm{T}}(4)$	0.00000	0.00000	0.00000
$r_\eta^{\mathrm{T}}(5)$	0.00000	0.00000	0.00000
$r_\eta^{\mathrm{T}}(6)$	0.00000	0.00000	0.00000
$r_\eta^{\mathrm{T}}(7)$	0.00000	0.00000	0.00000
$r_\eta^{\mathrm{T}}(8)$	0.00000	0.00000	0.00000
$r_\eta^{\mathrm{T}}(9)$	0.00000	0.00000	0.00000
$r_\eta^{\mathrm{T}}(10)$	0.00000	0.00000	0.00000

5.3 Выбор налучшей модели случайного процесса в классе моделей CC

Выбор лучшей модели проводится на основе анализа нормированных корреляционных функций с использованием критерия среднего квадратичного отклонения по первым десяти отсчётам (16).

Результаты представлены в таблице 5. По результатам делаем следующий вывод: среди данных моделей, процесс лучше всего описывает модель CC(1).

Таблица 5 – Результаты построения моделей CC(N)

Порядок модели	Параметры модели				Погрешность модели
M	α_0	α_1	α_2	α_3	ϵ^2
0	6.77840	-	-	-	0.791530
1	-2.70925	6.21343	-	-	0.657299
2	5.317996	-1.85165	3.77320	-	2.24461
3	Модель не существует				

6 ПОСТРОЕНИЕ И ИССЛЕДОВАНИЕ СМЕШАННЫХ МОДЕЛЕЙ АВТОРЕГРЕССИИ - СКОЛЬЗЯЩЕГО СРЕДНЕГО

6.1 Модель АРСС(M, N)

Модель АРСС(M, N) в общем виде может быть записана как:

$$\eta_n = \beta_1\eta_{n-1} + \beta_2\eta_{n-2} + \cdots + \beta_M\eta_{n-M} + \alpha_0\xi_n + \alpha_1\xi_{n-1} + \cdots + \alpha_N\xi_{n-N}, \tag{19}$$

где ξ – входная некоррелированная случайная последовательность с нулевым математическим ожиданием и единичной дисперсией ($\mathbb{M}(\xi_n) = 0, \mathbb{M}(\xi_n^2) = 1, \mathbb{M}(\xi_k\xi_n) = \delta_{kn}$), η – выходная случайная, стационарная в широком смысле последовательность с корреляционной функцией $R_\eta(m)$ и нулевым математическим ожиданием, $\{N, \alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_N, M, \beta_1, \beta_2, \ldots, \beta_M\}$ – параметры модели авторегрессии - скользящего среднего.

Для построения модели АРСС существует несколько методов, рассмотрим один из них.

6.2 Построение модели АРСС

Уравнение (19) можно переписать в виде:

$$\zeta_n = \alpha_0\xi_n + \alpha_1\xi_{n-1} + \cdots + \alpha_N\xi_{n-N}, \tag{20}$$

где

$$\zeta_n = \eta_n - \beta_1\eta_{n-1} - \beta_2\eta_{n-2} - \cdots - \beta_M\eta_{n-M}. \tag{21}$$

Уравнение (20) – уравнение модели СС(N) для промежуточной последовательности ζ, которая может быть получена из исходной последовательности η по уравнению (21).

Шаг 1. Из системы M линейных уравнений (22) находятся оценки M неизвестных коэффициентов $\beta_1, \beta_2, \ldots, \beta_M$.

$$\begin{cases} R_\eta(N+1) = \beta_1 R_\eta(N) + \beta_2 R_\eta(N-1) + \cdots + \beta_M R_\eta(N-M+1) \\ R_\eta(N+2) = \beta_1 R_\eta(N+1) + \beta_2 R_\eta(N) + \cdots + \beta_M R_\eta(N-M+2) \\ \qquad\qquad \cdots \\ R_\eta(N+M) = \beta_1 R_\eta(N+M-1) + \beta_2 R_\eta(N+M-2) + \cdots + \beta_M R_\eta(N) \end{cases} \tag{22}$$

Шаг 2. Из исходной последовательности η строится промежуточная последовательность ζ с использованием уравнения (21).

Шаг 3. Находится выборочная корреляционная функция $R_\zeta(m)$ промежуточной последовательности ζ.

Шаг 4. Из системы $N+1$ нелинейных уравнений (23) находится оценка $N+1$ неизвестных коэффициентов $\alpha_0, \alpha_1, \ldots, \alpha_N$.

$$\begin{cases} R_\eta(0) = \alpha_0^2 + \alpha_1^2 + \cdots + \alpha_N^2 \\ R_\eta(1) = \alpha_0\alpha_1 + \alpha_1\alpha_2 + \cdots + \alpha_{N-1}\alpha_N \\ \quad \cdots \\ R_\eta(N) = \alpha_0\alpha_N \end{cases} \tag{23}$$

Шаг 5. Проверить модель на устойчивость так же, как и в разделе (4.1). Выясним лежат ли все корни z характерестического уравнения (14) внутри единичной окружности $|z| < 1$ на комплексной плоскости.

Разберём построение модели АРСС(3, 1):

Шаг 1. Из системы трёх линейных уравнений (24) находим оценки неизвестных коэффициентов $\beta_1, \beta_2, \beta_3$.

$$\begin{cases} R_\eta(2) = \beta_1 R_\eta(1) + \beta_2 R_\eta(0) + \beta_3 R_\eta(-1) \\ R_\eta(3) = \beta_1 R_\eta(2) + \beta_2 R_\eta(1) + \beta_3 R_\eta(0) \\ R_\eta(4) = \beta_1 R_\eta(3) + \beta_2 R_\eta 2) + \beta_3 R_\eta(1) \end{cases} \tag{24}$$

Принимая во внимание что $R_\eta(-n) = R_\eta(n)$, получаем $\beta_1 = 0.19665, \beta_2 = 0.01413, \beta_3 = -0.80527$.

Шаг 2. Из исходной последовательности η строим промежуточную последовательность ζ с использованием уравнения (25).

$$\zeta_n = \eta_n - \beta_1\eta_{n-1} - \beta_2\eta_{n-2} - \beta_3\eta_{n-3}. \tag{25}$$

Шаг 3. Вычислим выборочную корреляционную функцию $R_\zeta(m)$ промежуточной последовательности ζ, используя формулу (6). Результаты представлены в таблице 6.

Таблица 6 – Выборочная корреляционная функция промежуточной последовательности ζ

m	$R_\zeta(m)$
0	63.45651
1	-0.88269
2	-1.07494
3	0.30147
4	0.53269
5	-1.97714
6	0.32706
7	1.06797
8	-0.62841
9	0.64616
10	1.30125

Шаг 4. Из системы двух нелинейных уравнений (26) находим оценку неизвестных коэффициентов α_0, α_1.

$$\begin{cases} R_\eta(0) = \alpha_0^2 + \alpha_1^2 \\ R_\eta(1) = \alpha_0\alpha_1 \end{cases} \tag{26}$$

Используем метод простых итераций из приложения Б.
Начальное приближение: $\alpha_0 = 0, \alpha_1 = \sqrt{R_\eta(0)}$.

Схема итераций:

1. $\alpha_0 \leftarrow \dfrac{R_\eta(1)}{\alpha_1}$.
2. Если $R_\eta(0) < \alpha_0^2$, то решений нет, иначе $\alpha_1 \leftarrow \sqrt{R_\eta(0) - \alpha_0^2}$.
3. Проверка на завершение итераций – уменьшение скорости сходимости итераций до заданного значения ϵ.
4. Переход к пункту 1.

В результате итерационного процесса было найденно одно из возможных решений ситсемы $\alpha_0 = -0.11082, \alpha_1 = 7.96519$, что является достаточным в данной работе.

Шаг 5. Проверим модель на устойчивость так же, как и в разделе (4.1).

Выясним лежат ли все корни z характерестического уравнения (14) внутри единичной окружности $|z| < 1$ на комплексной плоскости.

Модель APCC(3, 1) устойчива тогда и только тогда, когда $|\beta_3| < 1$, $|\beta_1 + \beta_3| < 1 - \beta_2$, $|\beta_2 + \beta_1\beta_3| < |1 - \beta_3^2|$:

$$\left.\begin{array}{l} |0.01413| < 1 \\ |0.21078| < 0.98587 \\ |0.01691| < 0.99980 \end{array}\right\} \Rightarrow \text{Модель устойчива.}$$

Используя данный метод построим оставшиеся модели APCC(M, N), для $M = 1, 2, 3$ и $N = 1, 2, 3$. Значения коэффициентов, полученных с помощью программы из приложения А, представлены в таблице 7.

6.3 Расчёт теоретической НКФ выходной последовательности

Для каждой модели рассчитаем теоретическую НКФ выходной последовательности. Для этого воспользуемся следующией формулой:

$$r_\eta^{\mathrm{T}}(N+M+k) = \beta_1 r_\eta^{\mathrm{T}}(N+M+k-1) + \beta_2 r_\eta^{\mathrm{T}}(N+M+k-2) + \cdots + \beta_M r_\eta^{\mathrm{T}}(N+k), k \geq 1, \quad (27)$$

учитывая что первые $N + M + 1$ значений теоретической НКФ нам уже известны, т.к. они совпадают с первыми значениями выборочной НКФ для исходных данных.

6.4 Выбор налучшей модели случайного процесса в классе моделей APCC

Выбор лучшей модели проводится на основе анализа нормированных корреляционных функций с использованием критерия среднего квадратичного отклонения по первым десяти отсчётам (16).

Результаты представлены в таблице 7. По результатам делаем следующий вывод: среди данных моделей, процесс лучше всего описывает модель APCC(3, 1).

Таблица 7 – Результаты построения моделей APCC(M, N)

Порядок модели		Параметры модели							Погрешность модели
M	N	β_1	β_2	β_3	α_0	α_1	α_2	α_3	ϵ^2
1	1	Модель не существует							
1	2	4.02560	-	-	Модель неустойчива				
1	3	Модель не существует							
2	1	2.22709	-1.02528	-	Модель неустойчива				
2	2	Модель не существует							
2	3	Модель не существует							
3	1	0.19665	0.01413	-0.80527	-0.11082	7.96519	-	-	0.00003
3	2	0.19232	0.02377	-0.80970	7.96346	-0.07843	-0.20571	-	0.00007
3	3	0.19009	0.02595	-0.81356	7.96320	-0.05910	-0.21906	0.08890	0.00006

7 СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПОСТРОЕННЫХ МОДЕЛЕЙ

В таблице 8 представлены погрешности моделей АРСС(M, N), для $M = 0, 1, 2, 3$ и $N = 0, 1, 2, 3$.

Таблица 8 – Погрешности моделей АРСС(M, N)

M \ N	0	1	2	3
0	2.37896	2.24461	-	-
1	2.41631	-	-	-
2	1.71611	-	-	-
3	0.00011	0.00003	0.00007	0.00006

Из таблицы видно, что исходный процесс лучше всего описывается моделями АР(3), СС(1) и АРСС(3,1). Смоделируем эти процессы принимая во внимание то, что сгенерированная случайная последовательность приобретает свойство стационарности по истичении интервала времени большего, чем интеравл корреляции. Поэтому сгенерируем 6000 отсчётов и отбросим первую 1000 элементов, считая их "браком". Код генерации представлен в приложении А.

Важно заметить то, что у смоделированного случайного процесса нулевое математическое ожидание. Чтобы исправить этот "недостаток"прибавим ко всем значениям математическое ожидание исходного процесса.

На рисунках 3, 4 и 5 изображены: выборочные НКФ исходного процесса, теоретические НКФ моделей и выборочные НКФ смоделированных процессов.

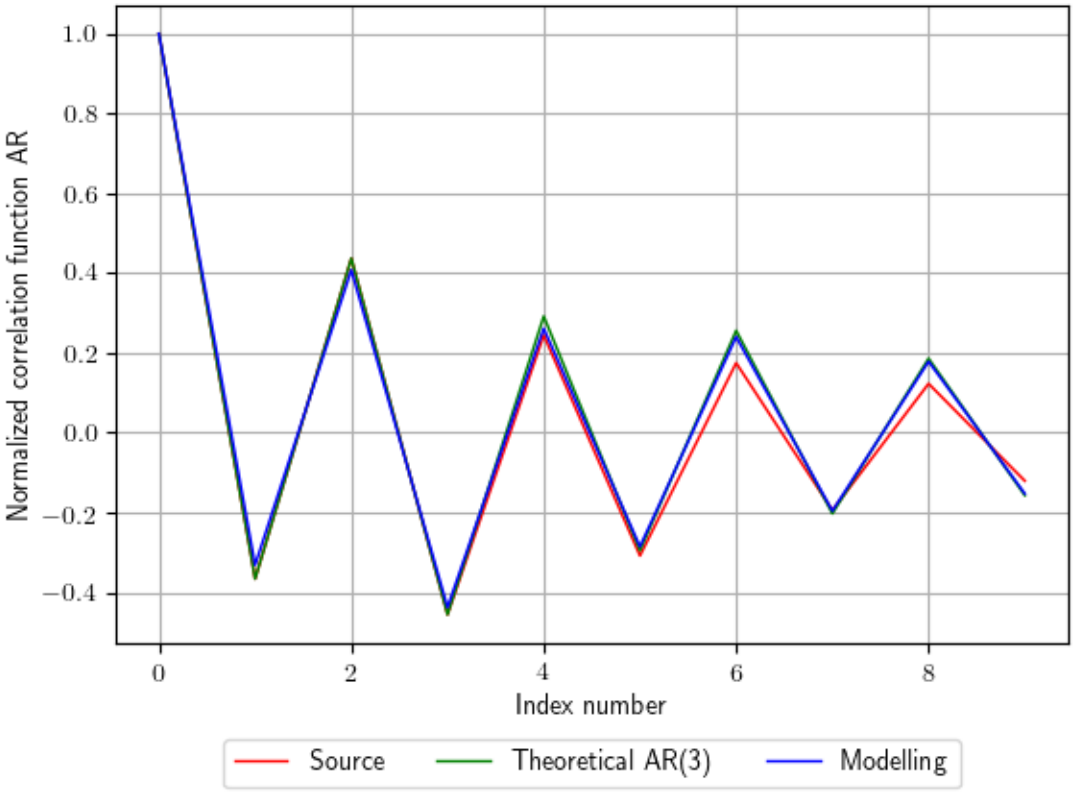

Рисунок 3 – Графическое сравнение НКФ для модели АР(3)

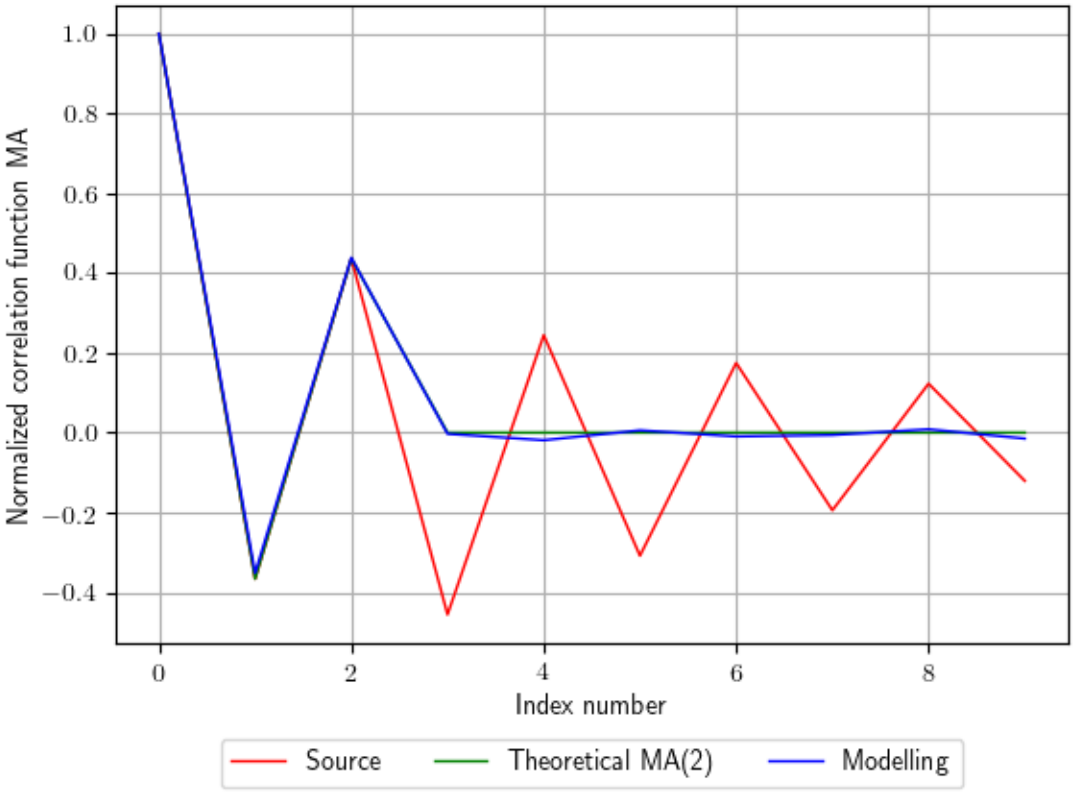

Рисунок 4 – Графическое сравнение НКФ для модели СС(1)

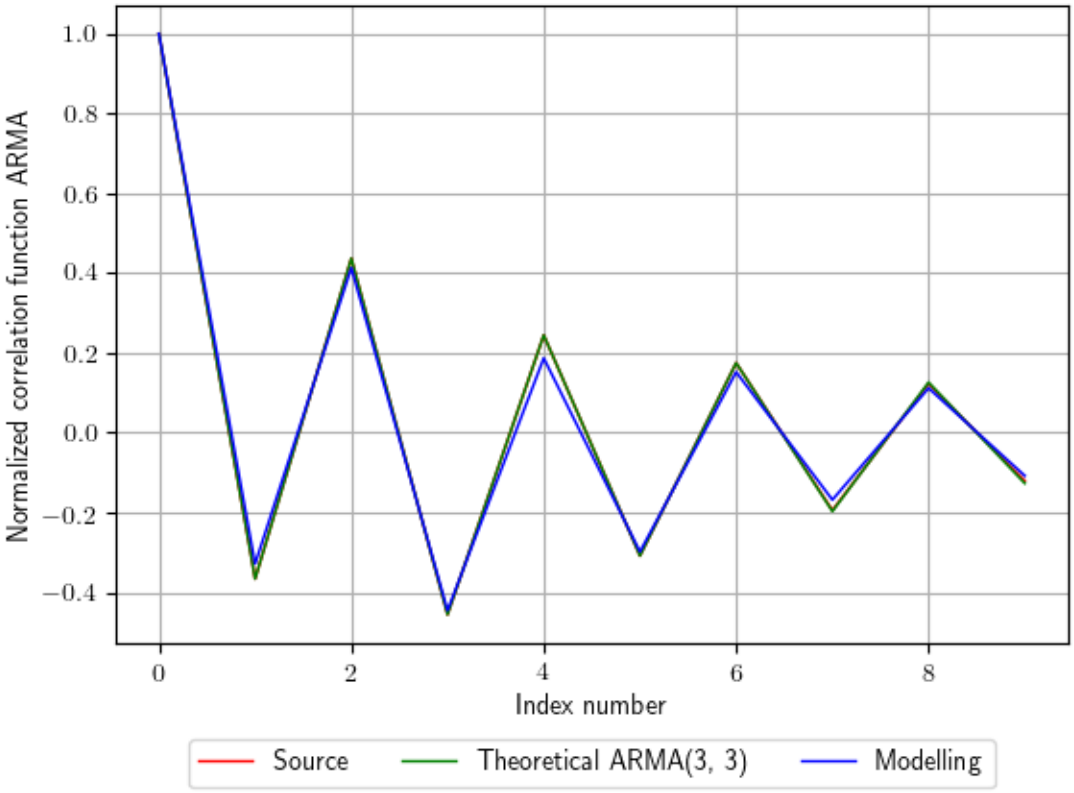

Рисунок 5 – Графическое сравнение НКФ для модели APCC(3, 1)

8 ИТОГОВАЯ ТАБЛИЦА СРАВНЕНИЯ МОДЕЛЕЙ АРСС

В таблице 9 представлены итоговые результаты.

Таблица 9 – Итоговые результаты статистического анализа СП и моделирования

Параметры процесса	Исходный процесс	АР(3) теория	АР(3) модель	СС(1) теория	СС(1) модель	АРСС(3,1) теория	АРСС(3,1) модель
Среднее	99.95990	99.95990	99.97495	99.95990	100.18219	99.95990	99.96754
Дисперсия	248.5539	248.5539	193.07658	248.5539	251.76423	248.5539	208.87360
СКО	15.76559	15.76559	13.89520	15.76559	15.86708	15.76559	14.45246
Нормированная корреляционная функция							
r(0)	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
r(1)	0.36654	0.36654	0.32207	0.36654	0.37392	0.36654	0.33774
r(2)	-0.20896	-0.20896	-0.17500	0.00000	-0.00312	-0.20896	-0.18725
r(3)	-0.84118	-0.84118	-0.80196	0.00000	-0.00569	-0.84118	-0.81092
r(4)	-0.46353	-0.46067	-0.38961	0.00000	-0.00685	-0.46353	-0.41711
r(5)	0.06005	0.06634	0.04976	0.00000	-0.00358	0.06523	0.06031
r(6)	0.68374	0.68406	0.62420	0.00000	-0.00655	0.68365	0.63791
r(7)	0.51005	0.50414	0.41071	0.00000	0.00354	0.50863	0.44453
r(8)	0.05844	0.0545	0.05103	0.00000	0.01235	0.05715	0.04911
r(9)	-0.53115	-0.53334	-0.46966	0.00000	0.00677	-0.53210	-0.47669
r(10)	-0.51157	-0.50832	-0.39948	0.00000	0.00273	-0.51341	-0.43103
Погрешность модели		0.00011	0.04005	2.24461	2.24212	0.00003	0.00349

Для нахождения погрешности использовался критерий среднего квадратичного отклонения по первым десяти отсчётам:

$$\epsilon^2 = \sum_{k=1}^{10} (r^{\mathrm{M}}(k) - \hat{r}(k))^2, \tag{28}$$

где $\hat{r}(k)$ – выборочная НКФ исходного процесса, r^{M} – выборочная НКФ смоделированного процесса.

На рисунке 6 представлен смоделированный процесс АРСС(3, 1).

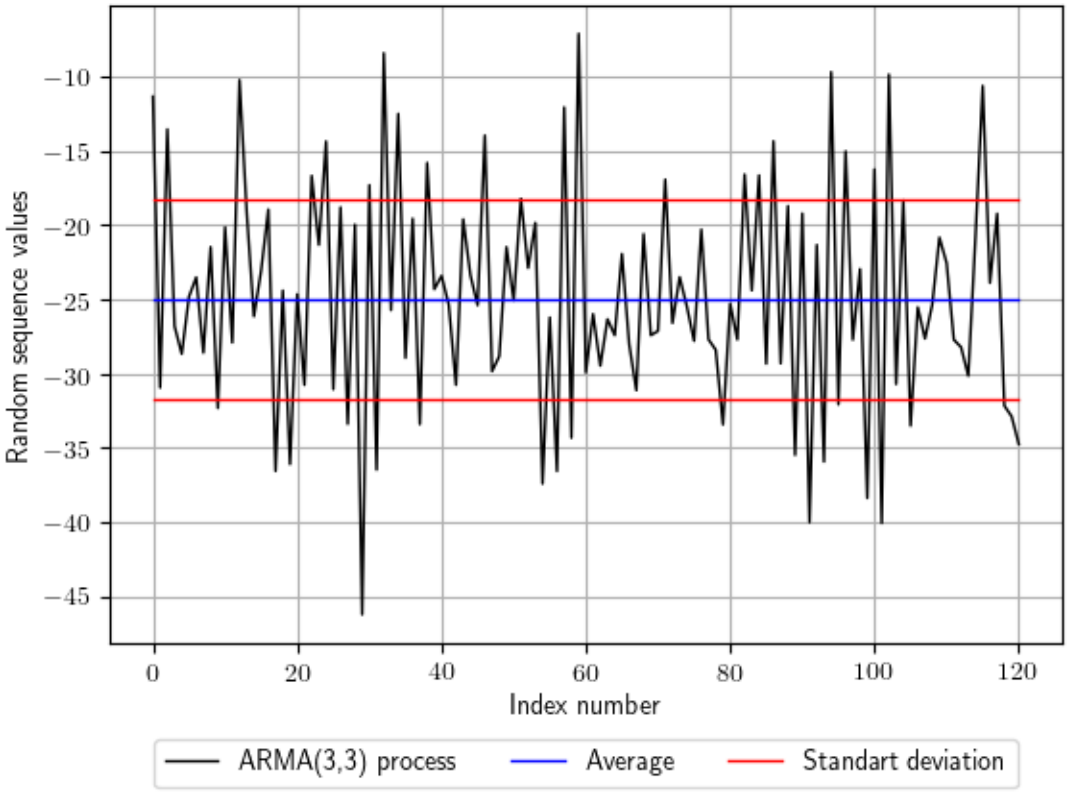

Рисунок 6 – Фрагмент реализации APCC(3, 1)

ЗАКЛЮЧЕНИЕ

Задача моделирования случайных процессов является довольно частой и важной. Модели авторегрессии и скользящего среднего позволяют моделировать случайные процессы, подобные исходному, по уже имеющейся реализации.

Было проведено исследование и моделирование для некоторого исходного неизвестного эргодического процесса. В ходе работы была проанализирована выборка из отсчётов исходного процесса, построены модели АР с помощью уравнений Юла-Уокера и модели СС с помощью систем нелинейных уравнений. Лучшими моделями в каждом классе оказались АР(3) и СС(1) с погрешностями 0.00011 и 2.24461 соответственно. Были построены смешанные модели АРСС до третьего порядка включительно, лучшей моделью среди них оказалась АРСС(3,1) с погрешностью равной 0.0001.

Для каждой из трёх лучших моделей были смоделированы случайные процессы, рассчитаны выборочные моментные функции, а также был произведен сравнительный анализ построенных моделей. Наилучшей оказалась модель АРСС(3,1) с погрегостью равной 0.00349.

СПИСОК ЛИТЕРАТУРЫ

1. **Тараскин, А.Ф.** Статистический анализ временных рядов авторегрессии и скользящего среднего: учебное пособие [Текст] // Самара: СГАУ, 1998. – 56с.

2. **Тараскин, А.Ф.** Статистическое моделирование и метод Монте–Карло: учебное пособие [Текст] // Самара: СГАУ, 1997. – 62с.

3. **Храмов, А.Г.** Анализ и моделирование процессов АРСС: интернет-ресурс к курсовой работе [Электронный ресурс] // Самара: СГАУ, 2009.

ПРИЛОЖЕНИЕ А КОД ПРОГРАММЫ

main.py:

```
from math import exp
import numpy as np
import matplotlib.pyplot as plt
from correlation import *
from moving_average import *
from autoregression import *
from arma import *

data = np.array(list(map(float, open('./tsp.txt'))))

if __name__ == '__main__':
    maxiter = 15000
    np.random.seed(0)

    mean = data.mean()
    std = data.std()

    print('Mean: {}'.format(mean))
    print('Disp.: {}'.format(std ** 2))
    print('STD: {}'.format(std))

    x = range(0, 121)
    y = data[0:121]

    fig = plt.figure()
    ax = plt.subplot(111)

    plt.rc('lines', linewidth=1)

    source, = ax.plot(x, y, 'black')
    avg, = ax.plot([0, 120], [mean] * 2, 'red')
    mn, = ax.plot([0, 120], [mean + std] * 2, 'blue')
    mn, = ax.plot([0, 120], [mean - std] * 2, 'blue')

    plt.xlabel('Index number')
    plt.ylabel('Random sequence values')

    box = ax.get_position()
    ax.set_position([box.x0, box.y0 + box.height * 0.1,
                     box.width, box.height * 0.9])
```

```
ax.legend([source, avg, mn], ['Source process', 'Average', 'Standart deviation'],
          loc='upper center', bbox_to_anchor=(0.5, -0.13), ncol=3, fancybox=True)

plt.grid(True)

plt.savefig('plot1.png')
plt.show()

cnst = exp(-1)
ncf = normalized_corr_func(data)
cf = corr_func(data)

limit = 300

for n in range(limit):
    ncf(n)

interval = limit

while abs(ncf(interval)) < exp(-1):
    interval -= 1

print('Correlation interval: {}'.format(interval))

print('Corr.f., ncf - first 10 values')
for n in range(11):
    print(n, ' & ', round(cf(n), 5), '& ', round(ncf(n), 5), ' \\\\ \\hline')

print('Corr.f, Norm.c.f in interval')
for n in range(interval + 1):
    print(round(cf(n), 5), round(ncf(n), 5))

fig = plt.figure()
ax = plt.subplot(111)

plt.rc('text', usetex=True)

x = range(interval + 20)
y = list(map(ncf, x))

source, = ax.plot(x, y, 'black')
cnst_plt, = ax.plot([min(x), max(x)], [cnst] * 2, 'blue')
```

```
cnst_plt, = ax.plot([min(x), max(x)], [-cnst] * 2, 'blue')
interval, = ax.plot([interval, interval], [
                 min(y), max(y)], 'red', linestyle='--')

plt.xlabel('Index number')
plt.ylabel('Normalized correlation function')

box = ax.get_position()
ax.set_position([box.x0, box.y0 + box.height * 0.1,
               box.width, box.height * 0.9])

ax.legend([source,
          cnst_plt,
          interval],
         ['Source',
          r'$\pm e^{-1}$',
          'Interval of correlation'],
         loc='upper center',
         bbox_to_anchor=(0.5,
                         -0.13),
         ncol=3,
         fancybox=True)

plt.grid(True)

plt.savefig('plot2.png')
plt.show()

# Autoregression
print('~' * 25)
print('Autoregression')
best_ar = 0
best_ar_err = 1e6
for m in range(4):
    try:
        print('Autoregression {}:'.format(m))
        print(' ' * 4 + str(calc_model_ar(m, data)))

        ncf_th = theory_ncf_ar(m, data)

        for x in range(1, 11):
            print(' ' * 4, x, round(ncf_th(x), 5))
```

```
        err = corr_error(ncf, ncf_th)
        print(' ' * 4 + str(err))

        if best_ar_err > err:
            best_ar = m
            best_ar_err = err
    except Exception as e:
        print(e)
print('Best ar err: {}, model ar({})'.format(best_ar_err, best_ar))

print('~' * 25)
print('Moving average')
best_ma = 0
best_ma_err = 1e6
for n in range(4):
    try:
        print('Moving average {}'.format(n))
        print(' ' * 4 + str(calc_model_ma(n, data, maxiter=maxiter)))

        ncf_th = theory_ncf_ma(n, data)

        for x in range(1, 11):
            print(' '*4, x, ncf_th(x))

        err = corr_error(ncf, ncf_th)
        print(' ' * 4 + str(err))

        if best_ma_err > err:
            best_ma = n
            best_ma_err = err
    except Exception as e:
        print(e)
print('Best ma err: {}, model ma({})'.format(best_ma_err, best_ma))

print('~' * 25)
print('ARMA')
best_arma_ar = 0
best_arma_ma = 0
best_arma_err = 1e6
for m in range(1, 4):
    for n in range(1, 4):
        try:
            print('ARMA({}, {})'.format(m, n))
```

```
            print(' ' * 4 + str(calc_model_arma(m, n, data, maxiter=maxiter)))

            ncf_th = theory_ncf_arma(m, n, data)

            for x in range(1, 11):
                print(' '*4, x, ncf_th(x))

            err = corr_error(ncf, ncf_th)
            print(' ' * 4 + str(err))

            if best_arma_err > err:
                best_arma_ar = m
                best_arma_ma = n
                best_arma_err = err
        except Exception as e:
            print(e)
print('Best arma err: {}, model arma({}, {})'.format(
    best_arma_err, best_arma_ar, best_arma_ma))

print('Data')
print('Mean: {}'.format(data.mean()))
print('Std: {}'.format(data.std()))

ar = model_ar(best_ar, data)([1] * best_ar, 1)
data_ar = np.array([next(ar) for _ in range(6000)][1000:]) + data.mean()

print('Modeled ar')
print('Mean: {}'.format(data_ar.mean()))
print('Mean err: {}'.format(abs(data_ar.mean() - data.mean())))
print('Std: {}'.format(data_ar.std()))
print('Std err: {}'.format(abs(data_ar.std() - data.std())))
ncf_ar = normalized_corr_func(data_ar)
for i in range(1, 11):
    print(i, round(ncf_ar(i), 5))
print('Error: {}'.format(round(corr_error(ncf, ncf_ar), 5)))

ma = model_ma(best_ma, data, maxiter=maxiter)(1)
data_ma = np.array([next(ma) for _ in range(6000)][1000:]) + data.mean()

print('Modeled ma')
print('Mean: {}'.format(data_ma.mean()))
print('Mean err: {}'.format(abs(data_ma.mean() - data.mean())))
print('Std: {}'.format(data_ma.std()))
```

```
print('Std err: {}'.format(abs(data_ma.std() - data.std())))
ncf_ma = normalized_corr_func(data_ma)
for i in range(1, 11):
    print(i, round(ncf_ma(i), 5))
print('Error: {}'.format(round(corr_error(ncf, ncf_ma), 5)))

arma = model_arma(
    best_arma_ar,
    best_arma_ma,
    data,
    maxiter=maxiter)(
    [1] *
    best_arma_ar,
    1)
data_arma = np.array([next(arma)
                      for _ in range(6000)][1000:]) + data.mean()

print('Modeled arma')
print('Mean: {}'.format(data_arma.mean()))
print('Mean err: {}'.format(abs(data_arma.mean() - data.mean())))
print('Std: {}'.format(data_arma.std()))
print('Std err: {}'.format(abs(data_arma.std() - data.std())))
ncf_arma = normalized_corr_func(data_arma)
for i in range(1, 11):
    print(i, round(ncf_arma(i), 5))
print('Error: {}'.format(round(corr_error(ncf, ncf_arma), 5)))

# AR plot

x = range(10)
y1 = list(map(ncf, x))
y2 = list(map(theory_ncf_ar(best_ar, data), x))
y3 = list(map(normalized_corr_func(data_ar), x))

fig = plt.figure()
ax = plt.subplot(111)

ncf_plot, = ax.plot(x, y1, 'red')
th_ncf_plot, = ax.plot(x, y2, 'green')
mdl_ncf_plot, = ax.plot(x, y3, 'blue')

plt.xlabel('Index number')
plt.ylabel('Normalized correlation function AR')
```

```
box = ax.get_position()
ax.set_position([box.x0, box.y0 + box.height * 0.1,
                 box.width, box.height * 0.9])

ax.legend([ncf_plot, th_ncf_plot, mdl_ncf_plot], ['Source', 'Theoretical AR({})'.format(
    best_ar), 'Modelling'], loc='upper center', bbox_to_anchor=(0.5, -0.13), ncol=3,
        ↪ fancybox=True)

plt.grid(True)

plt.savefig('plot_ar_ncf.png')
plt.show()

# MA plot

x = range(10)
y1 = list(map(ncf, x))
y2 = list(map(theory_ncf_ma(best_ma, data), x))
y3 = list(map(normalized_corr_func(data_ma), x))

fig = plt.figure()
ax = plt.subplot(111)

ncf_plot, = ax.plot(x, y1, 'red')
th_ncf_plot, = ax.plot(x, y2, 'green')
mdl_ncf_plot, = ax.plot(x, y3, 'blue')

plt.xlabel('Index number')
plt.ylabel('Normalized correlation function MA')

box = ax.get_position()
ax.set_position([box.x0, box.y0 + box.height * 0.1,
                 box.width, box.height * 0.9])

ax.legend([ncf_plot, th_ncf_plot, mdl_ncf_plot], ['Source', 'Theoretical MA({})'.format(
    best_ma), 'Modelling'], loc='upper center', bbox_to_anchor=(0.5, -0.13), ncol=3,
        ↪ fancybox=True)

plt.grid(True)

plt.savefig('plot_ma_ncf.png')
plt.show()
```

```
# ARMA plot

x = range(10)
y1 = list(map(ncf, x))
y2 = list(map(theory_ncf_arma(best_arma_ar, best_arma_ma, data), x))
y3 = list(map(normalized_corr_func(data_arma), x))

fig = plt.figure()
ax = plt.subplot(111)

ncf_plot, = ax.plot(x, y1, 'red')
th_ncf_plot, = ax.plot(x, y2, 'green')
mdl_ncf_plot, = ax.plot(x, y3, 'blue')

plt.xlabel('Index number')
plt.ylabel('Normalized correlation function ARMA')

box = ax.get_position()
ax.set_position([box.x0, box.y0 + box.height * 0.1,
                 box.width, box.height * 0.9])

ax.legend(
    [
        ncf_plot, th_ncf_plot, mdl_ncf_plot], [
        'Source', 'Theoretical ARMA({}, {})'.format(
            best_arma_ar, best_arma_ma), 'Modelling'], loc='upper center', bbox_to_anchor=(
                0.5, -0.13), ncol=3, fancybox=True)

plt.grid(True)

plt.savefig('plot_arma_ncf.png')
plt.show()

# best would be arma

x = range(0, 121)
y = data_arma[0:121]

mean = data_arma.mean()
std = data.std()

fig = plt.figure()
```

```
ax = plt.subplot(111)

data_plot, = ax.plot(x, y, 'black')
mean_plot, = ax.plot([0, 120], [mean, mean], 'blue')
std_plot, = ax.plot([0, 120], [mean - std, mean - std], 'red')
_ = ax.plot([0, 120], [mean + std, mean + std], 'red')

plt.xlabel('Index number')
plt.ylabel('Random sequence values')

box = ax.get_position()
ax.set_position([box.x0, box.y0 + box.height * 0.1,
                 box.width, box.height * 0.9])

ax.legend(
    [
        data_plot, mean_plot, std_plot], [
        'ARMA({},{}) process'.format(
            best_arma_ar, best_arma_ma), 'Average', 'Standart deviation'], loc='upper center
                ↪ ', bbox_to_anchor=(
                0.5, -0.13), ncol=3, fancybox=True)

plt.grid(True)

plt.savefig('plot_arma_modeled.png')
plt.show()
```

correlation.py:

```
import numpy as np
from functools import lru_cache


def corr_func(data):
    """
        Calculate correlation function of data
    """
    data = np.array(data)

    data = data - data.mean()

    @lru_cache(maxsize=None)
    def func(n):
        if n < 0:
            return func(-n)
        if n == 0:
            return data.std() ** 2

        x = data[:-n] * data[n:]

        return np.sum(x) / (np.size(x) - 1)

    return func


def normalized_corr_func(data):
    """
        Calculate normalized correlation function of data
    """
    data = np.array(data)

    corr = corr_func(data)

    dispersion = corr(0)

    @lru_cache(maxsize=None)
    def func(n):
        return corr(n) / dispersion

    return func
```

```python
def corr_error(corr1, corr2, n=11):
    return sum((corr1(n) - corr2(n))**2 for n in range(1, n))
```

moving_average.py:

```
import numpy as np
from correlation import *


def calc_model_ma0(data):
    """
        R(0) = a_0**2

        Exact solution:
        a_0 = sqrt(R(0))

        R - correlation function
        a_i - model coef.
    """
    data = np.array(data)

    R = corr_func(data)

    R0 = R(0)

    if R0 < 0:
        raise ArithmeticError(
            'Dispersion should be non-negative, maybe bug in corr_func')

    return np.sqrt([R0])


def theory_ncf_ma0(data):
    r = normalized_corr_func(data)

    def func(n):
        if n == 0:
            return r(0)
        return 0

    return func


def model_ma0(data):
    """
        x(t) = w(t) * a_0
```

```
        w - white noise
        a_i - coefs. from calc_model_ma0
    """
    a = calc_model_ma0(data)

    def get_model(x0):
        w = np.random.normal()
        x = x0

        while True:
            yield x

            x = w * a
            w = np.random.normal()

    return get_model


def calc_model_ma1(data, eps=1e-5, maxiter=100):
    """
        R(0) = a_0**2 + a_1**2
        R(1) = a_0 * a_1

        Iterative solution:
        a_0 = R(1) / a_1
        a_1 = sqrt(R(0) - a_0**2)

        R - correlation function
        a_i - model coefs.
    """
    data = np.array(data)

    R = corr_func(data)

    R0 = R(0)
    R1 = R(1)

    a0 = 0
    a1 = R0

    for _ in range(maxiter):
        if abs(R0 - a0**2 - a1**2) < eps \
               and abs(R1 - a0 * a1) < eps:
```

```
            return np.array([a0, a1])

        a0 = R1 / a1

        if (R0 - a0**2) < 0:
            raise ArithmeticError('Solution doesn\'t exist [ma(1)]')

        a1 = np.sqrt(R0 - a0**2)

    if abs(R0 - a0**2 - a1**2) > 10 * eps \
            or abs(R1 - a0 * a1) > 10 * eps:
        raise ArithmeticError('Didn\'t coverage to solution')

    return np.array([a0, a1])


def theory_ncf_ma1(data):
    r = normalized_corr_func(data)

    def func(n):
        if abs(n) <= 1:
            return r(n)
        return 0

    return func


def model_ma1(data, eps=1e-5, maxiter=100):
    """
        x(t) = a_0 * w(t) + a_1 * w(t - 1)

        w - white noise
        a_i - coefs. from calc_model_ma1
    """
    a = calc_model_ma1(data, eps, maxiter)

    def get_model(x0):
        w = np.random.normal(size=2)

        x = x0

        while True:
            yield x
```

```
            x = a.dot(w)

            w[1:] = w[:-1]
            w[0] = np.random.normal()

    return get_model


def calc_model_ma2(data, eps=1e-5, maxiter=100):
    """
        R(0) = a_0**2 + a_1**2 + a_2**2
        R(1) = a_0 * a_1 + a_1 * a_2
        R(2) = a_0 * a_2

        Iterative solution:
        a_2 = R(2) / a_0
        a_1 = (R(1) - a_1 * a_2) / a_0
        a_0 = sqrt(R(0) - a_1**2 - a_2**2) if R(0) >= a_1**2 + a_2**2

        R - correlation function
        a_i - model coeffs.
    """
    data = np.array(data)

    R = corr_func(data)

    R0 = R(0)
    R1 = R(1)
    R2 = R(2)

    a0 = np.sqrt(R0)
    a1 = 0
    a2 = 0

    for _ in range(maxiter):
        if abs(R0 - a0**2 - a1**2 - a2**2) < eps \
                and abs(R1 - a0 * a1 - a1 * a2) < eps \
                and abs(R2 - a0 * a2) < eps:
            return np.array([a0, a1, a2])

        a2 = R2 / a0
        a1 = (R1 - a1 * a2) / a0
```

```
        if R0 < a1 ** 2 + a2 ** 2:
            raise ArithmeticError('Solution doesn\'t exist [ma(2)]')

        a0 = np.sqrt(R(0) - a1**2 - a2**2)

    if abs(R0 - a0**2 - a1**2 - a2**2) > 10 * eps \
            or abs(R1 - a0 * a1 - a1 * a2) > 10 * eps \
            or abs(R2 - a0 * a2) > 10 * eps:
        raise ArithmeticError('Didn\'t coverage to solution')

    return np.array([a0, a1, a2])


def theory_ncf_ma2(data):
    r = normalized_corr_func(data)

    def func(n):
        if abs(n) <= 2:
            return r(n)
        return 0

    return func


def model_ma2(data, eps=1e-5, maxiter=100):
    """
        x(t) = a_0 * w(t) + a_1 * w(t - 1) + a_2 * w(t - 2)

        w - white noise
        a - coeffs. from calc_model_ma2
    """
    a = calc_model_ma2(data, eps, maxiter)

    def get_model(x0):
        x = x0
        w = np.random.normal(size=3)

        while True:
            yield x

            x = a.dot(w)
```

```
            w[1:] = w[:-1]
            w[0] = np.random.normal()

    return get_model


def calc_model_ma3(data, eps=1e-5, maxiter=100):
    """
        R(0) = a_0**2 + a_1**2 + a_2**2 + a_3**2
        R(1) = a_0 * a_1 + a_1 * a_2 + a_2 * a_3
        R(2) = a_0 * a_2 + a_1 * a_3
        R(3) = a_0 * a_3

        Iterative solution:
        a_3 = R(3) / a_0
        a_2 = (R(2) - a_1 * a_3) / a_0
        a_1 = (R(1) - a_1 * a_2 - a_2 * a_3) / a_0
        a_0 = sqrt(R(0) - a_1**2 - a_2**2 - a_3**2) if R(0) >= a_1**2 + a_2**2 + a_3**2

        R - correlation function
        a_i - model coeffs.
    """
    data = np.array(data)

    R = corr_func(data)

    R0 = R(0)
    R1 = R(1)
    R2 = R(2)
    R3 = R(3)

    a0 = np.sqrt(R0)
    a1 = 0
    a2 = 0
    a3 = 0

    for _ in range(maxiter):
        if abs(R0 - a0**2 - a1**2 - a2**2 - a3**2) < eps \
                and abs(R1 - a0 * a1 - a1 * a2 - a2 * a3) < eps \
                and abs(R2 - a0 * a2 - a1 * a3) < eps \
                and abs(R3 - a0 * a3) < eps:
            return np.array([a0, a1, a2, a3])
```

```
        a3 = R3 / a0
        a2 = (R2 - a1 * a3) / a0
        a1 = (R1 - a1 * a2 - a2 * a3) / a0

        if R0 < a1**2 + a2**2 + a3**2:
            raise ArithmeticError('Solution doesn\'t exist [ma(3)]')

        a0 = np.sqrt(R0 - a1**2 - a2**2 - a3**2)

    if abs(R0 - a0**2 - a1**2 - a2**2 - a3**2) > 10 * eps \
            or abs(R1 - a0 * a1 - a1 * a2 - a2 * a3) > 10 * eps \
            or abs(R2 - a0 * a2 - a1 * a3) > 10 * eps \
            or abs(R3 - a0 * a3) > 10 * eps:
        raise ArithmeticError('Didn\'t coverage to solution')

    return np.array([a0, a1, a2, a3])


def theory_ncf_ma3(data):
    r = normalized_corr_func(data)

    def func(n):
        if abs(n) <= 3:
            return r(n)
        return 0

    return func


def model_ma3(data, eps=1e-5, maxiter=100):
    """
        x(t) = a_0 * w(t) + a_1 * w(t - 1) + a_2 * w(t - 2) + a_3 * w(t - 3)

        w - white noise
        a - coeffs. from calc_model_ma3
    """
    a = calc_model_ma3(data, eps, maxiter)

    def get_model(x0):
        x = x0
        w = np.random.normal(size=4)

        while True:
```

```
            yield x

            x = a.dot(w)

            w[1:] = w[:-1]
            w[0] = np.random.normal()

    return get_model


def calc_model_ma(n, data, eps=1e-5, maxiter=100):
    """
        Calculate coeffs. for moving average model (ma(n)).
    """
    if n == 0:
        return calc_model_ma0(data)
    if n == 1:
        return calc_model_ma1(data, eps, maxiter)
    if n == 2:
        return calc_model_ma2(data, eps, maxiter)
    if n == 3:
        return calc_model_ma3(data, eps, maxiter)

    raise ValueError('Calculating coeffs. for `n > 3` not implemented')


def theory_ncf_ma(n, data):
    """
        Theoretical normalized correlation function for moving average model (ma(n)).
    """
    if n == 0:
        return theory_ncf_ma0(data)
    if n == 1:
        return theory_ncf_ma1(data)
    if n == 2:
        return theory_ncf_ma2(data)
    if n == 3:
        return theory_ncf_ma3(data)

    raise ValueError('NCF for `n > 3` not implemented')


def model_ma(n, data, eps=1e-5, maxiter=100):
```

```
    """
        Model of moving average (ma(n))
    """
    if n == 0:
        return model_ma0(data)
    if n == 1:
        return model_ma1(data, eps, maxiter)
    if n == 2:
        return model_ma2(data, eps, maxiter)
    if n == 3:
        return model_ma3(data, eps, maxiter)

    raise ValueError('Models with `n > 3` not implemented')
```

autoregression.py:

```
import numpy as np
from correlation import *


def solve(a, b):
    """
        Solves the linear system A * x = b
    """
    a = np.matrix(a)
    b = np.array(b)

    b = np.reshape(b, (-1, 1))

    x = np.linalg.solve(a, b)
    x = np.array(x)

    return x.flatten()


def make_walker_matrix(corr, n):
    """
        Returns matrix A for Yule-Walker system of range n
    """
    R = np.zeros((n, n))

    for i in range(n):
        for j in range(n):
            R[i, j] = corr(abs(i - j))

    return np.matrix(R)


def make_walker_vector(corr, n):
    """
        Returns vector b for Yule-Walker system of range n
    """
    R = np.zeros((n, 1))

    for i in range(n):
        R[i, 0] = corr(i + 1)

    return R
```

```
def calc_model_ar(n, data):
    """
        Calculates the autoregression model of range n with Yule-Walker equations
    """
    data = np.array(data)

    corr = corr_func(data)

    if n == 0:
        return np.sqrt([corr(0)])

    A = make_walker_matrix(corr, n)
    b = make_walker_vector(corr, n)

    betas = solve(A, b)

    tmp = corr(0) - betas.dot(b)

    if tmp < 0:
        raise ArithmeticError('No solution for model ar({})'.format(n))

    alpha = np.sqrt(corr(0) - betas.dot(b))

    """
        Checks the stability conditions for a given model
    """
    if n == 1:
        if abs(betas[0]) >= 1:
            raise ArithmeticError('Model ar(1) is unstable')

    if n == 2:
        if abs(betas[1]) >= 1 \
                or abs(betas[0]) >= 1 - betas[1]:
            raise ArithmeticError('Model ar(2) is unstable')

    if n == 3:
        if abs(betas[2]) >= 1 \
                or abs(betas[0] + betas[2]) >= 1 - betas[1] \
                or abs(betas[1] + betas[0] * betas[2]) >= abs(1 - betas[2]**2):
            raise ArithmeticError('Model ar(3) is unstable')
```

```
    return np.append(alpha, betas)


def theory_ncf_ar(n, data):
    coeffs = calc_model_ar(n, data)
    R = corr_func(data)

    @lru_cache(maxsize=None)
    def func(m):
        if abs(m) <= n:
            return R(m)

        if n == 0:
            return 0

        rs = np.array([func(m - x) for x in range(1, n + 1)])

        return rs.dot(coeffs[1:])

    def res(m):
        return func(m) / R(0)

    return res


def model_ar(n, data):
    coeffs = calc_model_ar(n, data)

    alpha = coeffs[0]

    if n > 0:
        betas = coeffs[1:]

    def get_model(xs, x0):
        if len(xs) != n:
            raise ValueError('Length of `xs` must be equal to {}'.format(n))

        w = np.random.normal()

        if n > 0:
            xs = np.array(xs)
            x = x0
```

```
    while True:
        yield x

        if n > 0:
            x = xs.dot(betas) + w * alpha

            xs[1:] = xs[:-1]
            xs[0] = x
        else:
            x = w * alpha

        w = np.random.normal()

return get_model
```

arma.py:

```
from correlation import *
from autoregression import *
from moving_average import *


def calc_model_arma(m, n, data, eps=1e-5, maxiter=100):
    """
    """
    data = np.array(data)

    matrix = np.zeros((m, m))
    vector = np.zeros((m, 1))

    R = corr_func(data)

    for i in range(m):
        vector[i] = R(n + i + 1)
        for j in range(m):
            matrix[i, j] = R(n + i - j)

    betas = solve(matrix, vector)

    def gen_seq():
        for i in range(m + 1, len(data)):
            xs = data[i - m - 1:i]
            y = xs[0] - xs[1:].dot(betas)

            yield y

    seq = gen_seq()
    seq_data = [x for x in seq]

    cf_seq = corr_func(seq_data)

    print('Start cf for temp. seq.')
    for i in range(11):
        print(i, cf_seq(i))
    print('End cf for temp. seq.')

    alphas = calc_model_ma(n, seq_data, eps, maxiter)

    if m == 1:
```

```
        if abs(betas[0]) >= 1:
            raise ArithmeticError('Solution unstable, betas: {}'.format(betas))
    if m == 2:
        if abs(betas[1]) >= 1 \
                or abs(betas[0]) >= 1 - betas[1]:
            raise ArithmeticError('Solution unstable, betas: {}'.format(betas))
    if m == 3:
        if abs(betas[2]) >= 1 \
                or abs(betas[0] + betas[2]) >= 1 - betas[1] \
                or abs(betas[1] + betas[0] * betas[2]) >= 1 - betas[2]**2:
            raise ArithmeticError('Solution unstable, betas: {}'.format(betas))

    return betas, alphas


def theory_ncf_arma(m, n, data, eps=1e-5, maxiter=100):
    """
    """
    from functools import lru_cache

    coeffs = calc_model_arma(m, n, data, eps, maxiter)

    ar_coeffs, ma_coeffs = coeffs

    R = corr_func(data)

    @lru_cache(maxsize=None)
    def func(x):
        if x < 0:
            return func(-x)
        if x <= m + n:
            return R(x)

        rs = np.array([func(x - i) for i in range(1, m + 1)])

        return rs.dot(ar_coeffs)

    def res(x):
        return func(x) / R(0)

    return res
```

```
def model_arma(m, n, data, eps=1e-5, maxiter=100):
    """
    """
    if m == 0:
        return model_ma(n, data, eps, maxiter)
    if n == 0:
        return model_ar(m, data)

    coeffs = calc_model_arma(m, n, data, eps, maxiter)

    ar_coeffs, ma_coeffs = coeffs

    def get_model(xs, x0):
        if len(xs) != m:
            raise ValueError('Length of `xs` must be equal to {}'.format(m))

        xs = np.array(xs)
        w = np.random.normal(size=(n + 1))

        x = x0

        while True:
            yield x

            x = xs.dot(ar_coeffs) + ma_coeffs.dot(w)

            w[1:] = w[:-1]
            w[0] = np.random.normal()

            xs[1:] = xs[:-1]
            xs[0] = x

    return get_model
```

ПРИЛОЖЕНИЕ Б МЕТОД ПРОСТЫХ ИТЕРАЦИИ

Итерационный метод решения системы нелинейных алгебраических уравнений рассмотрим на примере модели CC(2). Известно, что рассматриваемый ниже метод простых итераций для устойчивых моделей АРСС всегда позволяет найти одно из решений системы, если это решение существует.

Модель CC(2)

$$\eta_n = \alpha_0\xi_n + \alpha_1\xi_{n-1} + \alpha_2\xi_{n-2}$$

Система нелинейных уравнений для расчёта $\alpha_0, \alpha_1, \alpha_2$ имеет вид:

$$\begin{cases} R_\eta(0) = \alpha_0^2 + \alpha_1^2 + \alpha_2^2 \\ R_\eta(1) = \alpha_0\alpha_1 + \alpha_1\alpha_2 \\ R_\eta(2) = \alpha_0\alpha_2 \end{cases}$$

Значения $R_\eta(0), R_\eta(1), R_\eta(2)$ известны.

Начальное приближение: $\alpha_0 = \sqrt{R_\eta(0)}, \alpha_1 = 0, \alpha_2 = 0$.

Схема итераций:

1. $\alpha_2 \leftarrow \dfrac{R_\eta(2)}{\alpha_0}$.
2. $\alpha_1 \leftarrow \dfrac{R_\eta(1) - \alpha_1\alpha_2}{\alpha_0}$.
3. Если $R_\eta(0) < \alpha_1^2 + \alpha_2^2$, то решений нет, иначе $\alpha_0 \leftarrow \sqrt{R_\eta(0) - \alpha_1^2 - \alpha_2^2}$.
4. Проверка на завершение итераций – уменьшение скорости сходимости итераций до заданного значения ϵ.
5. Переход к пункту 1.

В результате итерационного процесса будет найденно одно из возможных решений ситсемы, что является достаточным в данной работе, или же $R_\eta(0) < \alpha_1^2 + \alpha_2^2$, что говорит о том, что решений нет.