**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П.Королёва»**

*ИНСТИТУТ ИНФОРМАТИКИ, МАТЕМАТИКИ И ЭЛЕКТРОНИКИ*

*ФАКУЛЬТЕТ ИНФОРМАТИКИ*

*КАФЕДРА ТЕХНИЧЕСКОЙ КИБЕРНЕТИКИ*

СТАТИСТИЧЕСКИЙ АНАЛИЗ И МОДЕЛИРОВАНИЕ

ПРОЦЕССОВ АВТОРЕГРЕССИИ И СКОЛЬЗЯЩЕГО СРЕДНЕГО

курсовая работа по дисциплине «Теория случайных процессов»

Вариант № 63

*Выполнил:* Лаптев А.В.

*Группа:* 6309

*№ зачётной книжки:* 146178

*Проверил:* Храмов А.Г.

*Оценка:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Дата:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Самара 2017

РЕФЕРАТ

Курсовая работа по курсу «Теория случайных процессов»

Курсовая работа 43 страницы, 15 таблиц, 6 рисунков, 1 приложение.

МОДЕЛИРОВАНИЕ СЛУЧАЙНЫХ ПРОЦЕССОВ, АВТОРЕГРЕССИЯ, СКОЛЬЗЯЩЕЕ СРЕДНЕЕ, СМЕШАННАЯ МОДЕЛЬ.

В данной курсовой работе проводится исследование выборки из отсчётов некоторого неизвестного стационарного эргодического случайного процесса и моделирование нового процесса, подобного исходному, с использованием моделей авторегрессии и скользящего среднего (АРСС) различных порядков. Модели АРСС исследуются на схожесть с исходным процессом, проводится построение графиков нормированных корреляционных функций для исходного и смоделированных процессов. Для наглядности большинство результатов изображено графически и в виде таблиц.

Программа и графики реализованы на интерпретируемом языке Python версии 3.5.3 с использованием библиотек NumPy, SciPy, MatPlotLib.

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:

24.977 2.496 24.642 7.087 11.618 2.476 0.198 23.670 10.873 8.175 20.945 11.274 -2.857 -12.181 32.655 25.919 0.059 -3.625 19.233 28.583

3.130 26.963 -5.814 9.652 27.839 18.366 -10.497 20.699 34.671 14.645

...

16.901 -0.330 -15.130 21.364 23.465

**ЗАДАНИЕ**

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

2. Построить модели авторегрессии АР(*M*) = АРСС (*M*, 0) порядков *M* = 1, 2, 3 (всего 3 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 наилучшую модель СП в классе моделей АР.

3. 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 наилучшую модель СП в классе моделей СС.

4. Построить смешанные модели авторегрессии – скользящего среднего (АРСС (*M*, *N*) до третьего порядка включительно (*M* = 1, 2, 3; *N* = 1, 2, 3) (всего 9 моделей). Рассчитать теоретические НКФ выходной последовательности для различных порядков моделей АРСС. На основе сравнения выборочной и теоретических НКФ выбрать наилучшую модель СП в классе смешанных моделей.

5. 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.

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

**СОДЕРЖАНИЕ**

[РЕФЕРАТ 2](#_Toc486519801)

[ИСХОДНЫЕ ДАННЫЕ 3](#_Toc486519802)

[ЗАДАНИЕ 4](#_Toc486519803)

[1 ОЦЕНКА МОМЕНТНЫХ ФУНКЦИЙ 7](#_Toc486519804)

[2 ПОСТРОЕНИЕ МОДЕЛЕЙ 11](#_Toc486519805)

[2.1 Общий вид модели АРСС 11](#_Toc486519806)

[2.2 Модели АР 11](#_Toc486519807)

[2.2.1 Модель АР(1) 12](#_Toc486519808)

[2.2.2 Модель АР(2) 13](#_Toc486519809)

[2.2.3 Модель АР(3) 14](#_Toc486519810)

[2.2.4 Расчет теоретической НКФ выходной последовательности 15](#_Toc486519811)

[2.2.5 Результаты построения моделей АР 19](#_Toc486519812)

[2.3 Модели СС 19](#_Toc486519813)

[2.3.1 Модель СС(0) 20](#_Toc486519814)

[2.3.2 Модель СС(1) 20](#_Toc486519815)

[2.3.3 Модель СС(2) 20](#_Toc486519816)

[2.3.4 Модель СС(3) 21](#_Toc486519817)

[2.3.5 Расчет теоретической НКФ выходной последовательности 21](#_Toc486519818)

[2.3.6 Результаты построения моделей CC 23](#_Toc486519819)

[2.4 Модели АРСС 24](#_Toc486519820)

[2.4.1 Примеры построения модели АРСС 25](#_Toc486519821)

[2.4.2 Результаты построения моделей АРCC 26](#_Toc486519822)

[3 МОДЕЛИРОВАНИЕ СЛУЧАЙНОГО ПРОЦЕССА 28](#_Toc486519823)

[3.1 Сравнение НКФ 29](#_Toc486519824)

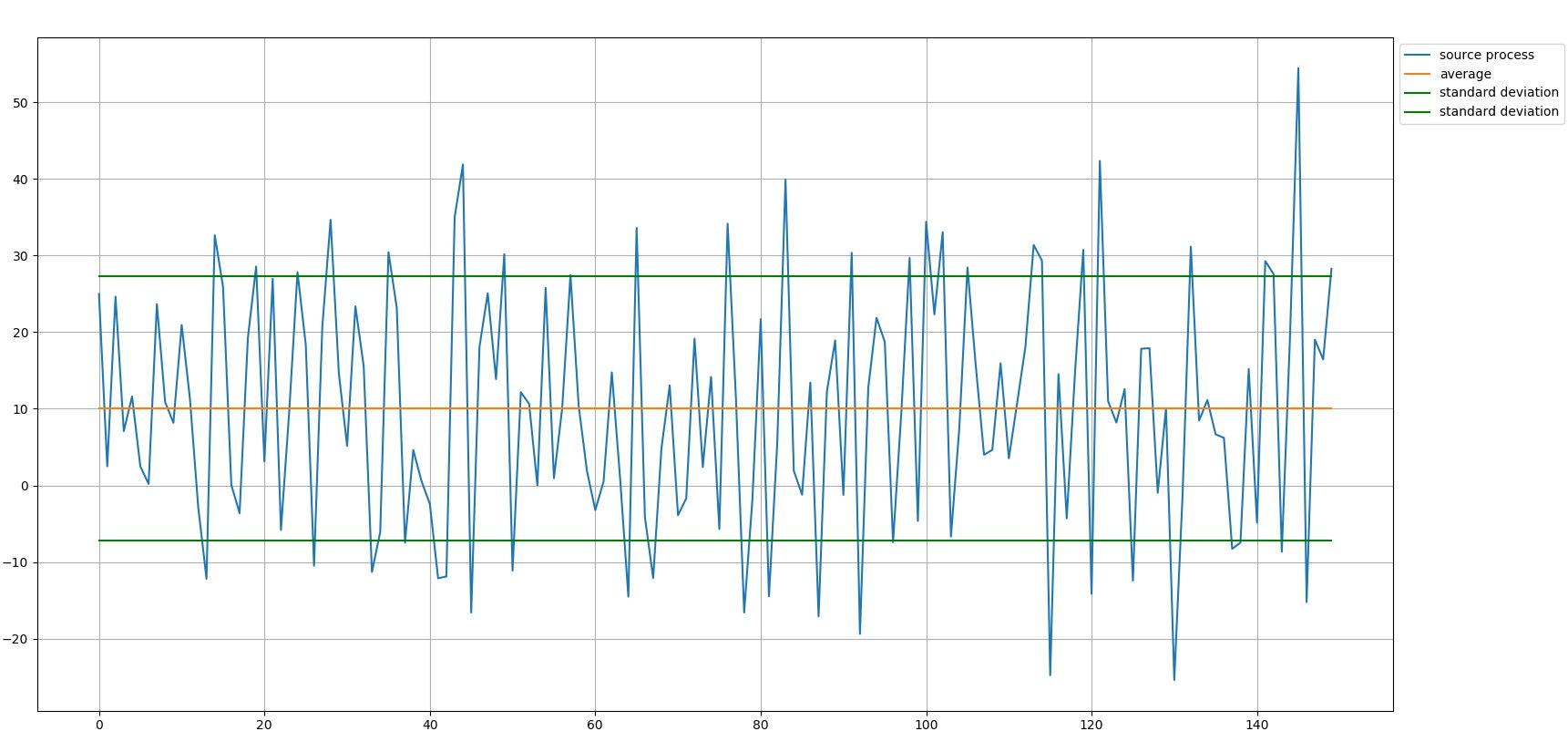
[3.2 Оценка моментных функций смоделированных процессов 31](#_Toc486519825)

[ЗАКЛЮЧЕНИЕ 33](#_Toc486519826)

[*ПРИЛОЖЕНИЕ А* Исходный код программы 34](#_Toc486519827)

1 ОЦЕНКА МОМЕНТНЫХ ФУНКЦИЙ

На рисунке 1 показаны первые 150 значений исходной выборки.



*Рисунок 1* – График фрагмента исходной выборки

Использовались библиотеки NumPy и Matplotlib для языка Python для расчетов и построения графиков соответственно.

Минимальное значение выборки *y* определено с помощью функции numpy.min(y).

Максимальное значение выборки *y* определено с помощью функции numpy.max(y).

Выборочное среднее значение рассчитано по формуле:

|  |  |
| --- | --- |
| , | (1.1) |

где – компонента вектора исходной выборки *y*, *n* – объем выборки, с помощью функции numpy.avg(y).

Получено значение выборочного среднего 10.04. Здесь и далее все данные расчётов приводятся с разумной степенью точности.

Выборочная дисперсия оценена по формуле исправленной выборочной дисперсии:

|  |  |
| --- | --- |
| , | (1.2) |

где – компонента вектора исходной выборки *y*, – выборочное среднее.

Получено значение выборочной дисперсии 297.8288.

Выборочное среднеквадратическое отклонение рассчитано по формуле .

Получено значение выборочного среднеквадратического отклонения 17.2577.

В таблице 1 представлена статистическая информация об исходной выборке.

*Таблица 1* – Статистическая информация об исходной выборке

|  |  |
| --- | --- |
| Минимальное значение | -47.408 |
| Максимальное значение | 67.562 |
| Выборочное среднее значение | 10.04 |
| Выборочная дисперсия | 297.8288 |
| Выборочное среднеквадратическое отклонение | 17.2578 |

Выборочная корреляционная функция оценена по формуле для расчета исправленной выборочной корреляционной функции:

|  |  |
| --- | --- |
|  | (1.3) |

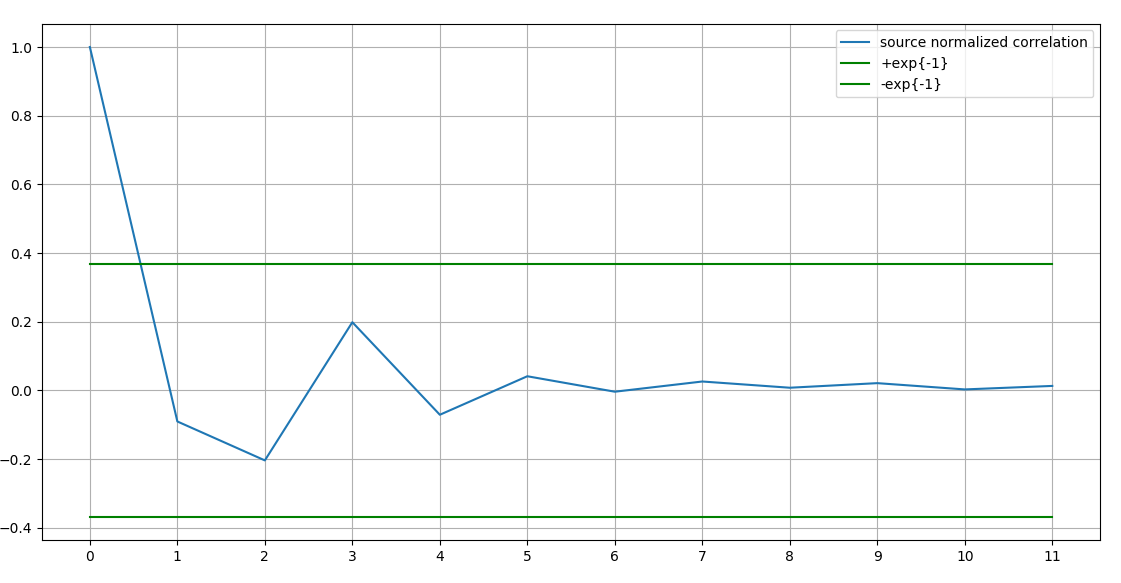
Выборочная нормированная корреляционная функция оценена по формуле для расчета исправленной выборочной нормированной корреляционной функции:

|  |  |
| --- | --- |
|  | (1.4) |

В таблице 2 показаны значения выборочной и выборочной нормированной корреляционных функций для . На рисунке 2 эти же значения изображены графически.

*Таблица 2* – Первые 11 значений выборочной нормированной корреляционной функции

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 0 | 297.8288 | 1 |
| 1 | -26.964 | -0.0905 |
| 2 | -60.7274 | -0.204 |
| 3 | 59.1316 | 0.199 |
| 4 | -21.1396 | -0.071 |
| 5 | 12.2567 | 0.0412 |
| 6 | -1.1477 | -0.0039 |
| 7 | 7.7003 | 0.0258 |
| 8 | 2.3009 | 0.0077 |
| 9 | 6.2760 | 0.02107 |
| 10 | 0.839 | 0.0028 |



*Рисунок 2* – График нормированной корреляционной функции

Интервал корреляции случайного процесса рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (1.5) |

Следует заметить, что вместо используется исправленная функция и что *m* не стоит полагать слишком большим относительно объема выборки во избежание чрезмерной ошибки.

Графически (по графику на рисунке 2) оценено значение интервала корреляции: 1.

2 ПОСТРОЕНИЕ МОДЕЛЕЙ

2.1 Общий вид модели АРСС

Общий вид модели авторегрессии и скользящего среднего (АРСС):

|  |  |
| --- | --- |
|  | (2.1) |

где – входная некоррелированная случайная последовательность с нулевым математическим ожиданием и единичной дисперсией, – выходная случайная последовательность с корреляционной функцией .

Модели АРСС (M, N) строится на основе решения системы уравнений Юла-Уокера следующим образом:

1. Отыщем коэффициенты из системы линейных уравнений:

|  |  |
| --- | --- |
|  | (2.2) |

1. Подставим в систему:

|  |  |
| --- | --- |
|  | (2.3) |

1. Найдем из следующей системы уравнений:

|  |  |
| --- | --- |
|  | (2.4) |

Найденные коэффициенты и описывают искомую модель.

2.2 Модели АР

Построим модели авторегрессии АР(*M*)= АРСС (*M*, 0) порядков *M*=1, 2, 3 на основе решения системы уравнений Юла-Уокера.

Общий вид модели авторегрессии представляет собой общий вид модели авторегрессии и скользящего среднего (2.1) при *N*=0:

|  |  |
| --- | --- |
|  | (2.5) |

Построим систему уравнений Юла-Уокера следующим образом:

1. Отыщем коэффициенты из системы линейных уравнений:

|  |  |
| --- | --- |
|  | (2.6) |

1. Подставим в систему:

|  |  |
| --- | --- |
|  | (2.7) |

1. Найдем из следующей системы уравнений:

|  |  |
| --- | --- |
|  | (2.8) |

Таким образом, система (2.7) принимает вид:

|  |  |
| --- | --- |
|  | (2.9) |

2.2.1 Модель АР (1)

При *M*=1 необходимо решить систему:

|  |  |
| --- | --- |
|  |  |

Решение найдено с использованием библиотеки NumPy. При замене система может быть представлена как система линейных уравнений относительно переменных и :

Для нахождения решения была использована функция numpy. linalg.solve(A, b). В результате работы функции получен вектор-решение:

Проверим найденную модель на устойчивость: модель устойчива.

2.2.2 Модель АР (2)

При *M*=2 необходимо решить систему:

|  |  |
| --- | --- |
|  |  |

Из свойства корреляционной функции стационарного в широком смысле процесса: .

Представим систему как систему линейных уравнений относительно переменных , и в матричном виде:

Получен вектор-решение:

Проверим систему на устойчивость: модель устойчива.

2.2.3 Модель АР (3)

При *M*=2 необходимо решить систему:

|  |  |
| --- | --- |
|  |  |

Представим систему как систему линейных уравнений относительно переменных , , и в матричном виде:

Получен вектор-решение:

Проверим систему на устойчивость: модель устойчива.

2.2.4 Расчет теоретической НКФ выходной последовательности

Для каждой модели рассчитываем теоретические нормированные корреляционные функции выходной последовательности. Недостающие значения корреляционной функции отыщем из системы:

|  |  |
| --- | --- |
|  | (2.10) |

в которой первые значений уже известны и совпадают со значениями выборочной корреляционной функции входной последовательности.

Затем найдем значения нормированной корреляционной функции по формуле:

|  |  |
| --- | --- |
|  | (2.11) |

Для АР (1) система выглядит следующим образом:

|  |  |
| --- | --- |
|  |  |

Значения нормированной корреляционной функции представлены в таблице 3.

*Таблица 3* – Первые 11 значений нормированной корреляционной функции для АР (1)

|  |  |
| --- | --- |
|  |  |
| 0 | 1 |
| 1 |  |
| 2 | 0.0819 |
| 3 | -0.0741 |
| 4 | 0.0671 |
| 5 | -0.0607 |
| 6 | 0.0549 |
| 7 | -0.0497 |
| 8 | 0.0450 |
| 9 | -0.0407 |
| 10 | 0.0369 |

Для АР (2) система выглядит следующим образом:

|  |  |
| --- | --- |
|  |  |

Значения нормированной корреляционной функции представлены в таблице 4.

*Таблица 4* – Первые 11 значений нормированной корреляционной функции для АР (2)

|  |  |
| --- | --- |
|  |  |
| 0 | 1 |
| 1 |  |
| 2 |  |
| 3 | 0,0418 |
| 4 | 0,0390 |
| 5 | -0,0132 |
| 6 | -0,0069 |
| 7 | 0,0036 |
| 8 | 0,0011 |
| 9 | -0,0009 |
| 10 | -0,0001 |

Для АР (3) система выглядит следующим образом:

|  |  |
| --- | --- |
|  |  |

Значения нормированной корреляционной функции представлены в таблице 5.

*Таблица 5* – Первые 11 значений нормированной корреляционной функции для АР (3)

|  |  |
| --- | --- |
|  |  |
| 0 | 1 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 | 0,0101 |
| 5 | -0,0735 |
| 6 | 0,0364 |
| 7 | 0,0133 |
| 8 | -0,0203 |
| 9 | 0,0049 |
| 10 | 0,0058 |

Выбор лучшей модели проводится на основании анализа рассчитанного теоретического среднеквадратичного отклонения нормированных корреляционных функций моделей от выборочной нормированной корреляционной функции по первым 11 отсчетам:

|  |  |
| --- | --- |
| где – рассчитанная теоретически нормированная корреляционная функция. | (2.12) |

По результату анализа лучшей моделью является АР (3).

2.2.5 Результаты построения моделей АР

Результаты построения модели представлены в таблице 3.

*Таблица 6* – Модели авторегрессии

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Порядок модели *M* | Параметры модели | | | | Погрешность модели |
|  |  |  |  |
| 1 | -0.0905 |  |  | 17.1868 | 0.0926 |
| 2 | -0.1099 | -0.2138 |  | 16.7893 | 0.0406 |
| 3 | -0.0745 | -0.1956 | 0.1656 | 16.5573 | **0.0225** |

Лучшей моделью будем считать АР (3) как обладающую наименьшей погрешностью.

2.3 Модели СС

Построим модели скользящего среднего СС(*N*) = АРСС (0, *N*) порядков *N* = 0, 1, 2, 3.

Общий вид модели скользящего среднего представляет собой общий вид модели авторегрессии и скользящего среднего (2.1) при *M*=0:

|  |  |  |  |
| --- | --- | --- | --- |
| Модель скользящего среднего устойчива всегда.  Расчеты будем производить исходя из значений корреляционной функции процесса и системы нелинейных уравнений:   |  |  | | --- | --- | |  | (2.14) |   Решение системы получено с помощью функции scipy.optimize.fsolve(func(x), x), имеющей два аргумента: func(x) и x. Эта функция находит решение системы нелинейных уравнений . В качестве аргумента func(x) выступает система (2.14). Она поступает как результат функции \_\_func(x), выдающей массив значений системы с помощью вложенного цикла. Внешний цикл реализует суммирование значений корреляционной функции с результатом работы внутреннего цикла. Внутренний цикл реализует суммирование произведений и по счетчику , при .  Второй аргумент функции scipy.optimize.fsolve – вектор x, являющийся начальным приближением вектора решений.  Функция scipy.optimize.fsolve возвращает вектор решений системы уравнений. | (2.13) |

2.3.1 Модель СС (0)

Для *N*=0 необходимо решить уравнение:

|  |  |
| --- | --- |
|  |  |

2.3.2 Модель СС (1)

Для *N*=1 необходимо решить систему нелинейных уравнений:

|  |  |
| --- | --- |
|  |  |

Система имеет два вещественных корня: -1.5689 и 17.1863, что означает существование модели с данными коэффициентами.

Для данной модели аргумент func(x) функции scipy.optimize.fsolve выглядит:

2.3.3 Модель СС (2)

Для *N*=2 необходимо решить систему нелинейных уравнений:

|  |  |
| --- | --- |
|  |  |

Система имеет три вещественных корня: -3.6262, -2.0551, 16.7468, что означает существование модели с данными коэффициентами.

2.3.4 Модель СС (3)

Для *N*=3 необходимо решить систему нелинейных уравнений:

Система имеет четыре вещественных корня: 11.3839, 7.8555, -8.9189, 5.1943, что означает существование модели с данными коэффициентами.

2.3.5 Расчет теоретической НКФ выходной последовательности

Для СС (0) НКФ рассчитывается из системы (2.14) делением корреляционной функции на выборочную дисперсию.

Так как существует все необходимые модели СС, то необходимо вычислить значения НКФ для всех моделей.

Значения нормированной корреляционной функции представлены в таблице 7.

*Таблица 7* – Первые 11 значений нормированной корреляционной функции для CC (0)

|  |  |
| --- | --- |
|  |  |
| 0 | 1 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |

Погрешность модели рассчитывается по формуле (2.12). Полученное значение погрешности: 0.0971.

*Таблица 8* – Первые 11 значений нормированной корреляционной функции для CC (1)

|  |  |
| --- | --- |
|  |  |
| 0 | 1 |
| 1 |  |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |

Погрешность модели рассчитывается по формуле (2.12). Полученное значение погрешности: 0.0889.

*Таблица 9* – Первые 11 значений нормированной корреляционной функции для CC (2)

|  |  |
| --- | --- |
|  |  |
| 0 | 1 |
| 1 |  |
| 2 |  |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |

Погрешность модели рассчитывается по формуле (2.12). Полученное значение погрешности: 0.0473.

*Таблица 10* – Первые 11 значений нормированной корреляционной функции для CC (3)

|  |  |
| --- | --- |
|  |  |
| 0 | 1 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 0 |
| 9 | 0 |
| 10 | 0 |

Погрешность модели рассчитывается по формуле (2.12). Полученное значение погрешности: **0.0079**.

2.3.6 Результаты построения моделей CC

Результаты построения модели представлены в таблице 11.

«Модель не существует» означает, что система уравнений для определения параметров модели не имеет решения.

Как видно, что существуют все четыре возможные модели. Модель СС (3) обладает наименьшей погрешностью. Ее будем считать лучшей.

*Таблица 11* – Модели скользящего среднего

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Порядок модели *M* | Параметры модели | | | | Погрешность модели |
|  |  |  |  |
| 0 | 17.2577 |  |  |  | 0.0971 |
| 1 | -1.5689 | 17.1863 |  |  | 0.0889 |
| 2 | -3.6262 | -2.0551 | 16.7468 |  | 0.0473 |
| 3 | 11.3839 | 7.8555 | -8.9189 | 5.1943 | **0.0079** |

2.4 Модели АРСС

Требуется построить модели АРСС (2.1) порядков *N*=1, 2, 3; *M*=1, 2, 3.

*Таблица 12* – Уравнения связей параметров модели АРСС(*M*, *N*) с корреляционной функцией

|  |  |  |
| --- | --- | --- |
| **Блок** | **Кол-во уравнений** | **Модель АРСС** |
| **А** | ***N*+1** | … |
| **Б** | ***M*** | … |
| **В** | **∞** | … |
| **Г** | ***N*+1** | … |

Оценивать параметры модели АРСС будем следующим методом:

1. Из уравнений блока **Б** в таблице 12 найдем оценки неизвестных коэффициентов .
2. Из уравнений блока **Г** в таблице 12 последовательно выражаются взаимные корреляционные функции и подставляются в уравнения блока **А**.
3. Из уравнений блока **А** в таблице 12 находятся оценки неизвестных коэффициентов .

2.4.1 Примеры построения модели АРСС

Для каждой из 9 моделей расписывать решение нецелесообразно, поэтому приведем примеры оценивания параметров АРСС (1, 1), АРСС (1,2) и АРСС (3, 1).

Вид модели при *M*=1, *N*=1:

1. Необходимо решить линейное уравнение:

|  |  |
| --- | --- |
|  |  |

Проверим найденную модель на устойчивость: модель неустойчива. Все последующие вычисления для этой модели не имеют практического смысла.

Вид модели при *M*=1, *N*=2:

1. Необходимо решить линейное уравнение:
2. Из уравнений блока **Г** выразим :

|  |  |
| --- | --- |
|  |  |

Подставим выраженные значения из предыдущей системы в уравнения блока **А**:

|  |  |
| --- | --- |
|  |  |

Решаем полученную систему относительно переменных и .

Вид модели при *M*=3, *N*=1:

1. Необходимо решить систему линейных уравнений:

|  |  |
| --- | --- |
|  |  |

Получим вектор-решение:

Проверим найденную модель на устойчивость: модель устойчива

Из уравнений блока **Г** выразим :

|  |  |
| --- | --- |
|  |  |

Подставим выраженные значения из предыдущей системы в уравнения блока **А**:

|  |  |
| --- | --- |
|  |  |

Решаем полученную систему относительно переменных и .

Для решения систем использовалась функция scipy.optimize.fsolve(func(x), x) из библиотеки SciPy.

2.4.2 Результаты построения моделей АРCC

Результаты построения модели представлены в таблице 13.

«Модель не существует» означает, что система уравнений для определения параметров модели не имеет решения. «Модель неустойчива» означает, что система уравнений для определения параметров модели имеет решение, но не выполняются условия устойчивости.

*Таблица 13* – Модели авторегрессии и скользящего среднего

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Порядок модели* | | *Параметры модели* | | | | | | |
| *M* | *N* |  |  |  |  |  |  |  |
| 1 | 1 |  |  |  | Модель неустойчива | | | |
| 1 | 2 | -0.9737 |  |  | 18.1027 | 13.302 | -4.8008 |  |
| 1 | 3 | -0.3575 |  |  | 13.6682 | 9.2723 | -6.0268 | 2.7378 |
| 2 | 1 | -0.8491 | -0.2807 |  | 16.1154 | 13.5485 |  |  |
| 2 | 2 | -0.7877 | -0.4189 |  | 16.4788 | 12.0601 | 2.5971 |  |
| 2 | 3 | -0.2285 | 0.1256 |  | 14.4502 | 6.3045 | -7.9165 | 3.366 |
| 3 | 1 | -0.5985 | -0.2582 | 0.0531 | 8.96633 | 16.4552 |  |  |
| 3 | 2 | 0.9079 | 1.0208 | 0.4761 | Модель неустойчива | | | |
| 3 | 3 | -0.0171 | 0.2921 | 0.0886 | 13.7762 | 4.4182 | -11.4563 | 2.8738 |

Рассчитаем значения теоретической НКФ по формуле (1.4), причем значения КФ необходимо вычислить из системы уравнений блока **В**.

Рассчитаем теоретические погрешности моделей по формуле (2.17). Результаты расчетов представлены в таблице 14.

*Таблица 14* – Теоретические погрешности моделей АРСС

|  |  |  |  |
| --- | --- | --- | --- |
| *M* | *N* | | |
| 1 | 2 | 3 |
| 1 |  | 0.1750 | 0.0018 |
| 2 | 0.0063 | 0.0084 | 0.0012 |
| 3 | 0.0067 |  | **0.0006** |

Наименьшей погрешностью обладает модель АРСС (3,3), ее будем считать наилучшей.

3 МОДЕЛИРОВАНИЕ СЛУЧАЙНОГО ПРОЦЕССА

Смоделируем случайный процесс с использованием наилучших моделей: АР (3), СС (3), АРСС (3,3).

Для АР (3):

Для СС (3):

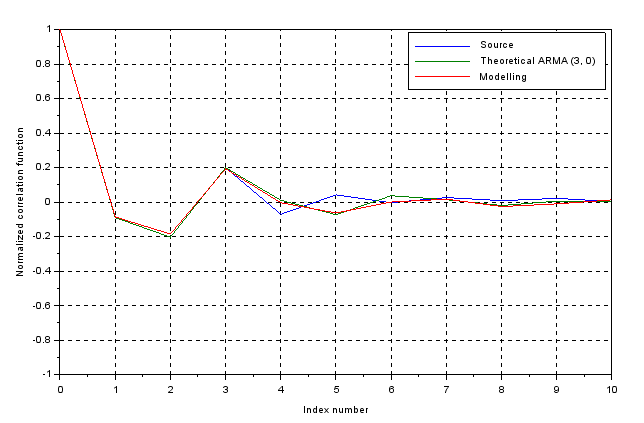
Для АРСС (3,3):

представляет собой нормальный вектор с нулевым математическим ожиданием и единичной дисперсией. Получим его с помощью функции numpy.random.normal(mu, sigma, size) и сместим на математическое ожидание исходного процесса, рассчитанное в разделе 1.

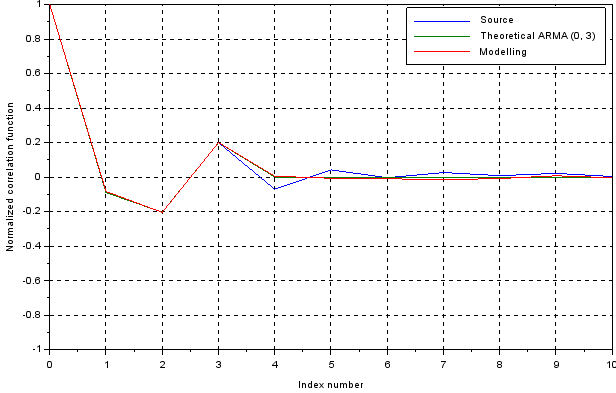
Следует учитывать, что при нулевых начальных условиях сгенерированная случайная последовательность приобретает свойство стационарности по истечению интервала времени, много большего, чем радиус корреляции. Поэтому сгенерируем 6000 отсчетов последовательности и отбросим первые 1000 отсчетов.

3.1 Сравнение НКФ

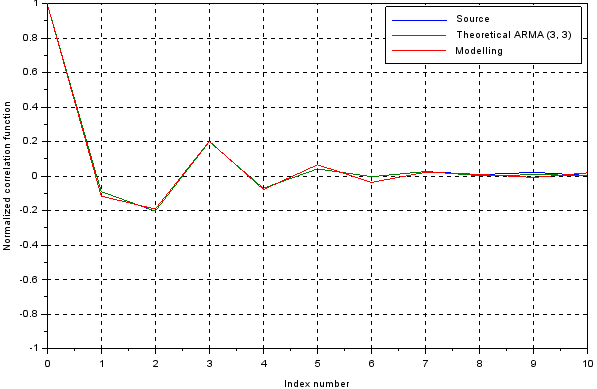
На рисунках 3-5а представлены графики выборочной НКФ исходного процесса, выборочной и теоретической НКФ смоделированных процессов.



*Рисунок 3* – Графики выборочной НКФ исходного случайного процесса, выборочной и теоретической НКФ смоделированного случайного процесса с использованием модели АР (3)



*Рисунок 4* – Графики выборочной НКФ исходного случайного процесса, выборочной и теоретической НКФ смоделированного случайного процесса с использованием модели СС (3)



*Рисунок 5а*– Графики выборочной НКФ исходного случайного процесса, выборочной и теоретической НКФ смоделированного случайного процесса с использованием модели АРСС (3,3)

3.2 Оценка моментных функций смоделированных процессов

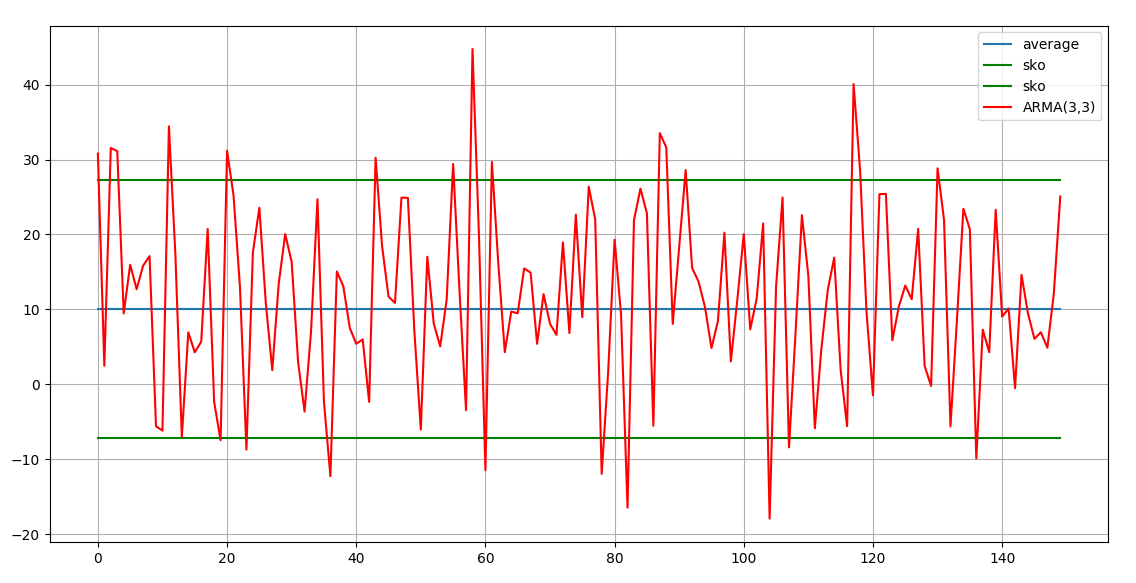
Построим оценки моментных функций смоделированных процессов и сравним их с оценками моментных функций исходного процесса и с теоретическими моментными функциями, соответствующими наилучшим моделям АР (3), СС (3) и АРСС (3,3). Расчёт выборочного среднего для лучших моделей происходит по формуле (1.1). Итоговые результаты статистического анализа и моделирования представлены в таблице 15.

*Таблица 15* – Результаты статистического анализа и моделирования

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Параметр** | **Исходный процесс** | **АР(3)** | | **СС(3)** | | **АРСС(3,3)** | |
| **Теория** | **Выборка** | **Теория** | **Выборка** | **Теория** | **Выборка** |
| Минимум | -47.408 | - | -79.708 | - | -78.82 | - | -73.941 |
| Максимум | 67.562 | - | 43.791 | - | 53.353 | - | 44.973 |
| Среднее | 10.04 | 10.04 | 9.600 | 10.04 | 9.8275 | 10.04 | 9.8238 |
| Дисперсия | 297.8288 | 297.8288 | 292.4873 | 297.8288 | 288.9517 | 297.8288 | 305.4006 |
| СКО | 17.2577 | 17.2577 | 17.1023 | 17.2577 | 16.9986 | 17.2577 | 17.4757 |
| Нормированная корреляционная функция | | | | | | | |
| r[0] | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| r[1] | -0.0905 | -0.0905 | -0.0867 | -0.0905 | -0.0838 | -0.0905 | -0.1165 |
| r[2] | -0.204 | -0.204 | -0.1858 |  | -0.2051 | -0.2039 | -0.1917 |
| r[3] | 0.199 | 0.199 | 0.1923 |  | 0.1995 | 0.1985 | 0.2020 |
| r[4] | -0.071 | -0.071 | -0.0044 | 0 | 0.0043 | -0.071 | -0.0789 |
| r[5] | 0.0412 | 0.0412 | -0.0637 | 0 | -0.0073 | 0.0412 | 0.0653 |
| r[6] | -0.0039 | -0.0039 | -0.0002 | 0 | -0.0101 | -0.0039 | -0.0373 |
| r[7] | 0.0258 | 0.0258 | 0.0163 | 0 | -0.0157 | 0.0259 | 0.0218 |
| r[8] | 0.0077 | 0.0077 | -0.0263 | 0 | -0.0090 | 0.0021 | 0.0093 |
| r[9] | 0.02107 | 0.02107 | -0.0109 | 0 | 0.0070 | 0.0071 | -0.0085 |
| r[10] | 0.0028 | 0.0028 | 0.0123 | 0 | -0.0041 | 0.0042 | 0.0184 |
| Погрешность | 0 | 0.0225 | 0.0252 | 0.0079 | 0.0104 | 0.0006 | 0.0037 |

Будем считать лучшей модель АРСС (3,3) как обладающую наименьшей погрешностью.

На рисунке 6 показаны первые 150 значений смоделированного с использованием модели АРСС (3,3) случайного процесса.



*Рисунок 6* – Результат моделирования случайного процесса с использованием модели АРСС (3,3)

ЗАКЛЮЧЕНИЕ

В работе была проанализирована выборка из отсчётов исходного процесса, построены модели АР с помощью уравнений Юла-Уокера, модель СС и смешанные модели АРСС с помощью систем нелинейных уравнений. Лучшими моделями в каждом классе оказались АР (3), СС (3) и АРСС (3,3). Как мы видим из таблицы, теоретические погрешности моделей СС (3) и АРСС (3,3) получились вполне точными, т.е. обладающими малой погрешностью. Лучшей моделью оказалась АРСС (3,3).

*ПРИЛОЖЕНИЕ А* Исходный код программы

*ar.py*

**import** numpy **as** np  
  
M = 0  
R = np.empty  
R\_th = np.empty  
coefs = np.empty  
  
**def** calc(m, r):  
 **global** M, R  
 M = m  
 R = r  
  
 \_\_calc\_coefs()  
  
 print(**'AR('**, M, **')'**)  
 print(**'coefs:'**, coefs)  
  
**def** \_\_calc\_coefs():  
 **global** coefs  
  
 A = np.zeros((M + 1, M + 1))  
 b = np.zeros(M + 1)  
  
 A[0, 0] = 1  
 i = 1  
 **while** i <= M:  
 A[0, i] = R[i]  
 i += 1  
 k = 1  
 b[0] = R[0]  
 **while** k <= M:  
 i = 1  
 **while** i <= M:  
 A[k, i] = R[np.absolute(k - i)]  
 i += 1  
 b[k] = R[k]  
 k += 1  
  
 coefs = np.linalg.solve(A, b)  
 coefs[0] = np.sqrt(coefs[0])  
  
**def** deviation(variance, r\_source):  
 **global** r, R\_th  
  
 R\_th = np.zeros(np.size(R))  
 k = 1  
 **while** k <= M:  
 R\_th[k-1] = R[k-1]  
 k += 1  
  
 **while** k <= np.size(R):  
 j = 1  
 **while** j <= M:  
 R\_th[k-1] += coefs[j]\*R\_th[k-j-1]  
 j += 1  
 k += 1  
  
 r = R\_th/variance  
eps = np.sum(np.power(r[1:np.size(r)-1] - r\_source[1:np.size(r\_source)-1], 2))  
 **return** eps  
  
**def** get\_model(M, offset):  
 calc(M, R)  
  
 e = np.random.normal(0, 1, 6000)  
 e = e[999:np.size(e)-1]  
  
 res = np.zeros(np.size(e))  
  
 n = 0  
 **while** n < np.size(e):  
 i = 1  
 m = M  
 **if** n < m:  
 m = n  
 **while** i <= m:  
 res[n] += coefs[i] \* res[n - i]  
 i += 1  
 res[n] += coefs[0] \* e[n]  
 n += 1  
  
 res += offset \* np.ones(np.size(e))  
 **return** res

*ma.py*

**import** scipy.optimize **as** solver  
**import** numpy **as** np  
**import** warnings  
  
warnings.filterwarnings(**'ignore'**, **'The iteration is not making good progress'**)  
  
N = 0  
R = np.empty  
equation\_exist = 0  
alphas = np.empty  
  
**def** calc(n, r):  
 **global** N, R, equation\_exist  
  
 N = n  
 R = r  
  
 \_\_calc\_alphas()  
  
 print(**'MA('**, N, **')'**)  
 f = \_\_func(alphas)  
 **if** f.max() < np.power(10.0, -3):  
 equation\_exist = 1  
 print(**'alphas:'**, alphas)  
 **else**:  
 equation\_exist = 0  
 print(**'Equation is not exist'**)  
  
**def** \_\_func (x):  
 out = np.ones(np.size(x))  
 k = 0  
 **while** k < np.size(x):  
 out[k] = -R[k]  
 i = 0  
 **while** i <= N-k:  
 out[k] += x[i]\*x[i+k]  
 i += 1  
 k += 1  
 **return** out  
  
**def** \_\_calc\_alphas():  
 **global** alphas  
  
 x = [1]  
 i = 1  
 **while** i <= N:  
 x.append(i + 1)  
 i += 1  
 alphas = solver.fsolve(\_\_func, x)  
  
**def** deviation(r\_source, variance):  
 **global** R  
  
 r = np.zeros(np.size(r\_source))  
  
 i = 0  
 **while** i <= N:  
 r[i] = R[i]/variance  
 i += 1  
  
 eps = np.sum(np.power(r[1:np.size(r)-1] - r\_source[1:np.size(r\_source)-1], 2))  
 **return** eps  
  
**def** get\_model(N, offset):  
 calc(N, R)  
  
 e = np.random.normal(0, 1, 6000)  
 e = e[999:np.size(e)-1]  
  
 res = np.zeros(np.size(e))  
  
 res[0] = alphas[0]\*e[0]  
 n = 1  
 **while** n < np.size(e):  
 i = 0  
 **while** i <= N:  
 res[n] += alphas[i]\*e[n-i]  
 i += 1  
 n += 1  
  
 res += offset \* np.ones(np.size(e))  
 **return** res

*arma.py*

**import** numpy **as** np  
**import** scipy.optimize **as** solver  
  
R = np.empty  
r = np.empty  
Re = np.empty  
betas = np.empty  
alphas = np.empty  
M = 0  
N = 0  
data = np.empty  
equation\_exist = 0  
  
**def** calc(m, n, r):  
 **global** R, M, N, betas, equation\_exist  
  
 R = r  
 M = m  
 N = n  
  
 \_\_calc\_betas()  
 \_\_calc\_alphas()  
  
 print(**'ARMA('**,M,**','**,N,**')'**)  
 f = \_\_func(alphas)  
 **if** f.max() < np.power(10.0, -3):  
 equation\_exist = 1  
 print(**'alphas:'**, alphas)  
 print(**'betas:'**, betas)  
 **else**:  
 equation\_exist = 0  
 print(**'Equation is not exist'**)  
  
**def** \_\_calc\_betas():  
 **global** R, M, N, betas  
  
 A = np.zeros((M, M))  
 b = np.zeros(M)  
  
 k = N + 1  
 i = 0  
 **while** k <= N + M:  
 m = 1  
 j = 0  
 **while** m <= M:  
 A[i, j] = R[k-m]  
 m += 1  
 j += 1  
 b[i] = R[k]  
 k += 1  
 i += 1  
  
 betas = np.linalg.solve(A, b)  
  
**def** \_\_func(x):  
 **global** Re  
  
 Re = np.zeros(N + 1)  
  
 k = 0  
 **while** k <= N:  
 i = 1  
 m = k  
 **if** m > M:  
 m = M  
 **while** i <= m:  
 Re[k] += betas[i - 1] \* Re[np.absolute(k - i)]  
 i += 1  
 Re[k] += x[k]  
 k += 1  
  
 out = np.zeros(np.size(x))  
 k = 0  
 **while** k < np.size(x):  
 out[k] = -R[k]  
 i = 1  
 **while** i <= M:  
 out[k] += betas[i-1]\*R[np.absolute(k-i)]  
 i += 1  
 j = k  
 **while** j <= N:  
 out[k] += x[j]\*Re[np.absolute(j-k)]  
 j += 1  
 k += 1  
 **return** out  
  
**def** \_\_calc\_alphas():  
 **global** alphas  
  
 x = [1]  
 i = 1  
 **while** i <= N:  
 x.append(i + 1)  
 i += 1  
  
 alphas = solver.fsolve(\_\_func, x)  
  
**def** deviation(r\_source):  
 **global** r  
  
 r = np.zeros(np.size(r\_source))  
 i = 0  
 **while** i <= M+N:  
 r[i] = r\_source[i]  
 i += 1  
  
 **while** i < np.size(r\_source):  
 j = 1  
  
 **while** j <= M:  
 r[i] += betas[j-1]\*r\_source[np.absolute(i-j)]  
 j += 1  
  
 i += 1  
  
 eps = np.sum(np.power(r[1:np.size(r) - 1] - r\_source[1:np.size(r\_source) - 1], 2))  
 **return** eps  
  
**def** get\_model(M, N, offset):  
 calc(M, N, R)  
  
 e = np.random.normal(0, 1, 6000)  
 e = e[999:np.size(e)-1]  
  
 res = np.zeros(np.size(e))  
  
 n = 0  
 **while** n < np.size(e):  
 i = 1  
 m = M  
 **if** n < m:  
 m = n  
 **while** i < m:  
 res[n] += betas[i-1] \* res[n - i]  
 i += 1  
 j = 0  
 **while** j <= N:  
 res[n] += alphas[j]\*e[n-j]  
 j += 1  
 n += 1  
  
 res += offset \* np.ones(np.size(e))  
 **return** res

*functions.py*

**import** numpy **as** np  
  
**def** correlation\_function(x, n, k, average):  
 q = 1/(n-k-1)  
  
 i = 0  
 sum = 0  
 **while** i < n-k:  
 sum += (x[i] - average)\*(x[i+k]-average)  
 i += 1  
  
 res = q \* sum  
 **return** res

*main.py*

**import** models.arma **as** arma  
**import** models.ar **as** ar  
**import** models.ma **as** ma  
**import** matplotlib.pyplot **as** mplt  
**import** numpy **as** np  
**import** functions  
  
y = np.fromfile(**'data.txt'**, dtype=float, sep=**' '**)  
n = y.size  
x = np.linspace(0, n - 1, n)  
  
min = np.min(y)  
max = np.max(y)  
average = np.average(y)  
variance = np.sum(np.power(y - average, 2))/(n-1)  
standard\_deviation = np.sqrt(variance)  
  
print(**'min ='**, min)  
print(**'max ='**, max)  
print(**'average ='**, average)  
print(**'variance ='**, variance)  
print(**'standard deviation ='**, standard\_deviation)  
  
*#show plot of source process*mplt.plot(x[0:150], y[0:150], label=**'source process'**)  
mplt.plot(x[0:150], np.full(150, average), label=**'average'**)  
mplt.plot(x[0:150], np.full(150, average + standard\_deviation), **'g'**,  
 np.full(150, average - standard\_deviation), **'g'**,  
 label=**'standard deviation'**)  
mplt.legend(bbox\_to\_anchor=(1, 1))  
mplt.grid()  
mplt.show()  
  
k = 0  
m = 30  
R = np.zeros(m)  
r = np.zeros(m)  
  
**while** k <= m-1:  
 R[k] = functions.correlation\_function(y, n, k, average)  
 r[k] = R[k] / variance  
 k += 1  
  
eps = np.exp(-1)  
*#show plot of source normalized correlation*mplt.plot(np.linspace(0, m - 1, m), r, label=**'source normalized correlation'**)  
mplt.plot(np.linspace(0, m - 1, m), np.full(m, eps), **'g'**, label=**'+exp{-1}'**)  
mplt.plot(np.linspace(0, m - 1, m), np.full(m, -eps), **'g'**, label=**'-exp{-1}'**)  
mplt.xticks(np.linspace(0, m - 1, m))  
mplt.legend(bbox\_to\_anchor=(1, 1))  
mplt.grid()  
mplt.show()  
  
m = 11  
  
r\_source = np.zeros(m+1)  
i = 0  
**while** i <= m:  
 r\_source[i] = r[i]  
 i += 1  
  
R = R[0:m]  
r = r[0:m]  
  
M = 1  
**while** M <= 3:  
 ar.calc(M, R)  
 eps = ar.deviation(variance, r)  
 print(**'eps ='**, eps)  
 M += 1  
  
N = 0  
**while** N <= 3:  
 ma.calc(N, R)  
 **if** ma.equation\_exist == 1:  
 eps = ma.deviation(r, variance)  
 print(**'eps ='**, eps)  
 N += 1  
  
M = 1  
**while** M <= 3:  
 N = 1  
 **while** N <= 3:  
 arma.calc(M, N, R)  
 **if** arma.equation\_exist == 1:  
 eps = arma.deviation(r)  
 print(**'eps ='**, eps)  
 N += 1  
 M += 1  
  
ma\_model = ma.get\_model(0, average)  
mplt.plot(x[0:150], y[0:150], label=**'source process'**)  
mplt.plot(x[0:150], np.full(150, average), label=**'average'**)  
mplt.plot(x[0:150], np.full(150, average + standard\_deviation), **'g'**,  
 np.full(150, average - standard\_deviation), **'g'**,  
 label=**'standard deviation'**)  
mplt.plot(x[0:150], ma\_model[0:150], **'r'**, label=**'MA(0)'**)  
mplt.legend(bbox\_to\_anchor=(1, 1))  
mplt.grid()  
mplt.show()  
  
min = np.min(ma\_model)  
max = np.max(ma\_model)  
average = np.average(ma\_model)  
variance = np.sum(np.power(ma\_model - average, 2))/(n-1)  
standard\_deviation = np.sqrt(variance)  
  
print(**'min ='**, min)  
print(**'max ='**, max)  
print(**'average ='**, average)  
print(**'variance ='**, variance)  
print(**'standard deviation ='**, standard\_deviation)  
  
k = 0  
m = 10  
n = 15  
  
**while** k <= m:  
 R[k] = functions.correlation\_function(ma\_model, n, k, average)  
 r[k] = R[k] / variance  
 print(**'r('**,k,**') ='**,r[k])  
 k += 1  
  
eps = np.sum(np.power(r[1:np.size(r) - 1] - r\_source[1:np.size(r) - 1], 2))  
print(**'eps ='**, eps)  
  
ar\_model = ar.get\_model(3, average)  
mplt.plot(x[0:150], y[0:150], label=**'source process'**)  
mplt.plot(x[0:150], np.full(150, average), label=**'average'**)  
mplt.plot(x[0:150], np.full(150, average + standard\_deviation), **'g'**,  
 np.full(150, average - standard\_deviation), **'g'**,  
 label=**'standard deviation'**)  
mplt.plot(x[0:150], ar\_model[0:150], **'r'**, label=**'AR(3)'**)  
mplt.legend(bbox\_to\_anchor=(1, 1))  
mplt.grid()  
mplt.show()  
  
min = np.min(ar\_model)  
max = np.max(ar\_model)  
average = np.average(ar\_model)  
variance = np.sum(np.power(ar\_model - average, 2))/(n-1)  
standard\_deviation = np.sqrt(variance)  
  
print(**'min ='**, min)  
print(**'max ='**, max)  
print(**'average ='**, average)  
print(**'variance ='**, variance)  
print(**'standard deviation ='**, standard\_deviation)  
  
k = 0  
**while** k <= m:  
 R[k] = functions.correlation\_function(ar\_model, n, k, average)  
 r[k] = R[k] / variance  
 print(**'r('**,k,**') ='**,r[k])  
 k += 1  
  
eps = np.sum(np.power(r[1:np.size(r)-1] - r\_source[1:np.size(r)-1], 2))  
print(**'eps ='**,eps)  
  
arma\_model = arma.get\_model(3, 3, average)  
mplt.plot(x[0:150], y[0:150], label=**'source process'**)  
mplt.plot(x[0:150], np.full(150, average), label=**'average'**)  
mplt.plot(x[0:150], np.full(150, average + standard\_deviation), **'g'**,  
 np.full(150, average - standard\_deviation), **'g'**,  
 label=**'standard deviation'**)  
mplt.plot(x[0:150], arma\_model[0:150], **'r'**, label=**'ARMA(3,3)'**)  
mplt.legend(bbox\_to\_anchor=(1, 1))  
mplt.grid()  
mplt.show()  
  
min = np.min(arma\_model)  
max = np.max(arma\_model)  
average = np.average(arma\_model)  
variance = np.sum(np.power(arma\_model - average, 2))/(n-1)  
standard\_deviation = np.sqrt(variance)  
  
print(**'min ='**, min)  
print(**'max ='**, max)  
print(**'average ='**, average)  
print(**'variance ='**, variance)  
print(**'standard deviation ='**, standard\_deviation)  
  
k = 0  
**while** k <= m:  
 R[k] = functions.correlation\_function(arma\_model, n, k, average)  
 r[k] = R[k] / variance  
 print(**'r('**,k,**') ='**,r[k])  
 k += 1  
  
eps = np.sum(np.power(r[1:np.size(r)-1] - r\_source[1:np.size(r)-1], 2))  
print(**'eps ='**,eps)