САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Кафедра систем управления и информатики

Отчет по лабораторной работе №2

по дисциплине «Системное программное обеспечение»

2 вариант

Выполнили: студенты гр. R3335

Ал-Наим Рами

Костровская Ольга

Носкова Полина

Преподаватель:

Капитонов А.А.

Санкт-Петербург

2018 г.

Метод Наименьших Квадратов

Задание:

1) Реализовать алгоритм, заполняющий таблицу неповторяющимися координатами x и y. Количество координат n равно квадратному корню, из номера варианта помноженному на 10 и округленному в большую сторону. Диапазон значений координат вводится пользователем при запуске программы. Для четных вариантов таблица формируется в Excel или другом оффлайновом аналоге. Для нечетных вариантов таблица формируется в таблицах google.

2) Для заданных координат реализовать алгоритм метода наименьших квадратов (не используя готовые библиотеки для МНК) и построить график (библиотека matplotlib).

Решение:

Первая часть:

Создаем функцию, которая позволит пользователю задавать диапазон значений для 15 вводимых координат.

```
def start():
        limits = input("Укажите максимальное и минимальное значение случайной величины\n(Например: 0, 20):\n")

        min_val, max_val = limits.split(", ")
        min_val = float(min_val)
        max_val = float(max_val)

        initTable(min_val, max_val)
```

Создаем таблицу в Excel. Записываем в нее значения 15 координат с помощью функции random.

```
def initTable(minim=-20, maxim=20):

        wb = Workbook()
        ws = wb.active

        upper_limit = maxim
        lower_limit = minim

        coords = set()

        ws['A1'] = 'x'
        ws['A2'] = 'y'

        for col in range(2, 17):
                unique_coords = False
                while not unique_coords:
                        x_y = (random.uniform(lower_limit, upper_limit), random.uniform(lower_limit, upper_limit))
                        if x_y not in coords:
                                unique_coords = True
                                coords.add(x_y)
                                ws.cell(column=col, row=1, value=x_y[0])
                                ws.cell(column=col, row=2, value=x_y[1])


        wb.save("Data.xlsx")
```

Вторая часть:

Реализуем алгоритм метода наименьших квадратов. Постараемся понять, что это такое.

Метод наименьших квадратов (МНК, англ. Ordinary Least Squares, OLS) — математический метод, применяемый для решения различных задач, основанный на минимизации суммы квадратов отклонений некоторых функций от искомых переменных.

$$F(a,b) = \sum_{i=1}^{n} (y_i - (a \cdot x_i + b))^2 \to min$$

Мы будем использовать его для решения нашей задачи: построение аппроксимирующей прямой так, чтобы экспериментальные точки лежали максимально близко к ней.

Нам нужно реализовать следующие формулы, полученные в результате нахождения экстремума указанной функции двух переменных (F(a,b)).

$$sumx = \sum_{i=1}^{n} x_i \qquad sumy = \sum_{i=1}^{n} y_i$$

$$sumx2 = \sum_{i=1}^{n} x_i^2 \qquad sumxy = \sum_{i=1}^{n} x_i \cdot y_i$$

$$\begin{cases} a = \dfrac{n \cdot sumxy - sumx \cdot sumy}{n \cdot sumx2 - sumx^2} \\ b = \dfrac{sumy - a \cdot sumx}{n} \end{cases}$$

Как это выглядит на python:

```
sumx = 0
sumx2 = 0
sumy = 0
sumxy = 0

x_coords = []
y_coords = []

for point in data:

        plt.scatter(point[0], point[1])

        sumx += point[0]
        sumy += point[1]

        sumxy += point[0] * point[1]
        sumx2 += point[0] ** 2

        x_coords.append( point[0] )


a = ( n * sumxy - sumx * sumy ) / ( n * sumx2 - sumx ** 2 )
b = 1.0 * ( sumy - a * sumx ) / n
```

В итоге мы получили линейную функцию – аппроксимирующую прямую. Можно увидеть, что все экспериментальные точки лежали наиболее близко к аппроксимирующей прямой. Делаем вывод, что наш метод был успешно проверен в ходе экспериментальной реализации программы на python.

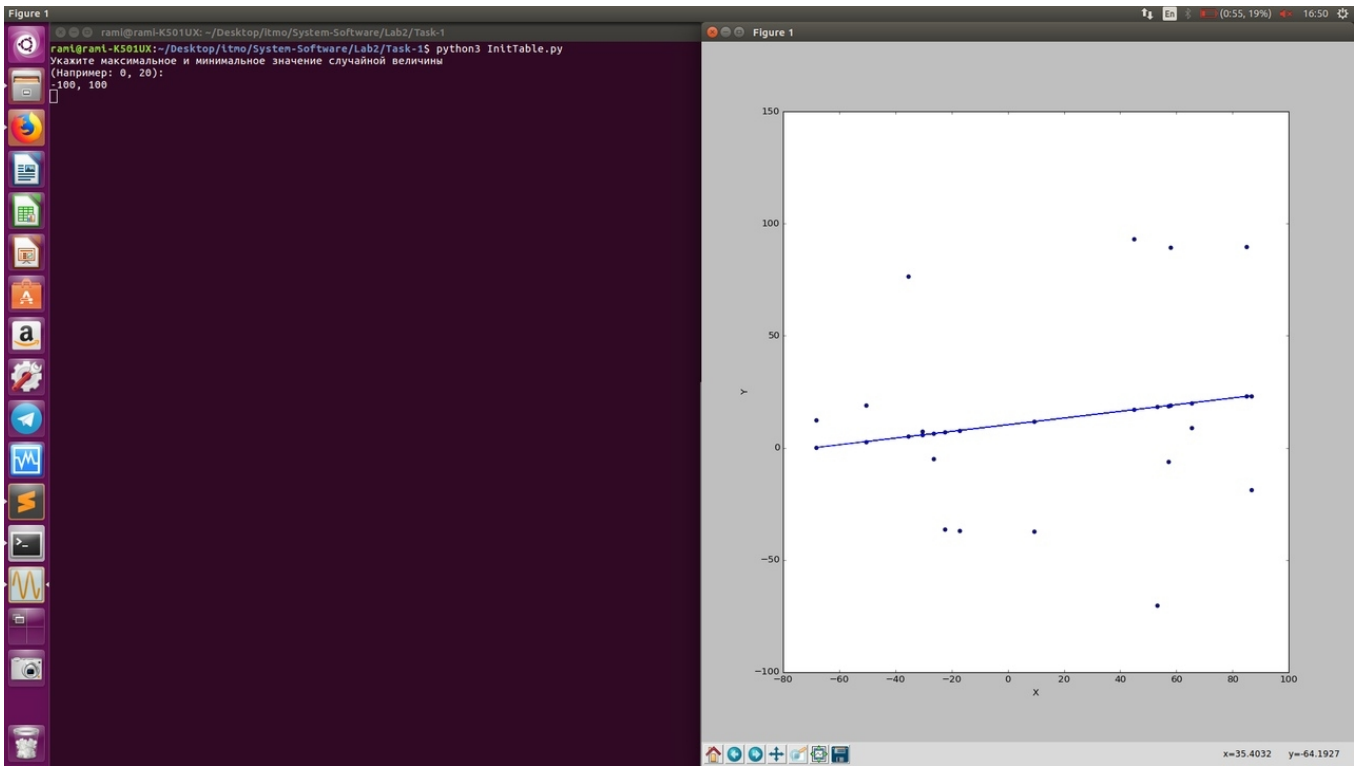

Рис. 1: Метод наименьших квадратов

Восстановление данных

Задание:

1) Создать таблицу nxn, заполненную случайными величинами в диапазоне от 1 до 30. Удалить значения из 10 случайных ячеек.

2) Реализовать алгоритм, восстанавливающий данные путем винзорирования.

3) Реализовать алгоритм, восстанавливающий данные путем линейной аппроксимации.

4) Реализовать алгоритм, восстанавливающий значения путем корреляционного восстановления. Коррелируемые между собой ряды измерений выбирает пользователь при запуске программы.

5) Проанализировать достоинства и недостатки методов восстановления данных.

Решение:

Первая часть:

Создаем таблицу 15x15, заполняем ее с помощью функции random в пределах от 1 до 30.

Так же с помощью random удаляем 10 случайных ячеек в диапазоне от 1 до 15 по столбцам и от 1 до 15 по строкам.

```
def initTable(wb, ws):
	data = set()
	for i in range(1,16):
		for j in range(1,16):
			val = random.uniform(1,30)
			ws.cell(column=j, row=i, value=val)
	val = 0
	i = 0
	while i < 10 :
		col = random.randint(1, 15)
		row = random.randint(1, 15)
		ws.cell(column=col, row=row, value=val)
		i+=1
	wb.save("Data1.xlsx")
```

Вторая часть:

Реализуем алгоритм, восстанавливающий данные путем винзорирования.

Если отсутствует число в первой клетке первого столбца нашей таблицы, то записываем в нее число 15 (среднее значение 1 и 30). Если отсутствует значение во второй, третьей, …, пятнадцатой клетке первого столбца, то записываем в нее значение из клетки, находящейся в последнем столбце и одной строкой выше. Если отсутствует число в любой другой клетке, то записываем в нее значение из соседней левой клетки.

```
def vinz(wb, ws):
	for i in range(1,16):
		for j in range(1,16):
			if ws['A1'].value == 0:
				ws.cell(column=1, row=1, value=15)
				continue
			if j==1 and ws[get_column_letter(j) + str(i)].value == 0:
				val = ws['O' + str(i-1)].value
				ws.cell(column=j, row=i, value=val)
				continue
			if ws[get_column_letter(j) + str(i)].value == 0:
				val = ws[get_column_letter(j-1) + str(i)].value
				ws.cell(column=j, row=i, value=val)
	wb.save("Data1.xlsx")
```

Третья часть:

Реализуем алгоритм, восстанавливающий данные путем линейной аппроксимации.

Используем уже разобранный алгоритм – метод наименьших квадратов. Строим аппроксимирующую прямую так, чтобы экспериментальные точки лежали максимально близко к ней. Линейная аппроксимация реализуется по столбцам.

```
def linapr(wb, ws):
	for j in range(1,16):
		sumx = 0
		sumx2 = 0
		sumy = 0
		sumxy = 0
		for i in range(1,16):
			sumx += i-1
			sumx2 += (i-1)**2
			sumy += ws[get_column_letter(j) + str(i)].value
			sumxy += ws[get_column_letter(j) + str(i)].value*(i-1)
		a = (15*sumxy - sumx*sumy)/(15*sumx2 - sumx**2)
		b = (sumy-a*sumx)/15
		for i in range(1,16):
			if ws[get_column_letter(j) + str(i)].value == 0:
				val = a*i + b
				ws.cell(column=j, row=i, value=val)
	wb.save("Data1.xlsx")
```

Четвертая часть:

Реализуем алгоритм, восстанавливающий значения путем корреляционного восстановления.

Выбираем два коррелируемых между собой столбца, например, 1 и 2. Если отсутствует значение X1, то восстанавливаем его по формуле X1=Y1/(X2*Y2). Поскольку пропущенный элемент находится в самом левом столбце, то мы составляем пропорцию с элементами из соседнего правого столбца. Если пропущенный элемент находится не в самом левом столбце, то мы составляем пропорцию с элементами из соседнего левого столбца.

X1	X2
Y1	Y2

```
def cor(wb, ws):
        n_str = input("Введите номер ряда, который вы хотите восстановить:\n")
        n = int(n_str)
        m_str = input("Введите номер ряда, с которым он коррелирует:\n")
        m = int(m_str)
        for j in range(1,16):
                if ws[get_column_letter(j) + str(n)].value == 0 and j==1:
                        val = ws[get_column_letter(j+1) + str(n)].value/(ws[get_column_letter(j) + str(m)].value*ws[get_column_letter(j+1) + str(m)].value)
                        ws.cell(column=j, row=n, value=val)
                elif ws[get_column_letter(j) + str(n)].value == 0:
                        val = ws[get_column_letter(j-1) + str(n)].value/(ws[get_column_letter(j) + str(m)].value*ws[get_column_letter(j-1) + str(m)].value)
                        ws.cell(column=j, row=n, value=val)
        wb.save("Data1.xlsx")
```

Пятая часть:

Проанализируем достоинства и недостатки методов восстановления данных.

Первый метод – винзорирования оказался самым простым в реализации. Алгоритм работает так, что он не пропустит ни одну клетку с удаленным значением. При этом у этого метода самая большая погрешность, основной минус метода.

Второй метод – метод линейной аппроксимации. Алгоритм чуть более сложен в реализации, требуются владеть математическим аппаратом. При большом количестве аномальных значений на месте пропуска может оказаться значение с большим смещением. Основной минус этого метода – большая погрешность при аномальных данных. Если изначально данные стремятся к образованию линейной зависимости, то погрешность будет небольшая.

Третий метод – корреляционное восстановление. Это самый сложный алгоритм из трех. При очень неудачном случайном удалении данных (например, если удалены подряд 10 нулей) не все данные удастся восстановить. Но в целом, более усложненная формула для отсутствующих элементов (в сравнении с винзонированием, где вместо пустого значения вписывается соседнее число) позволяет снизить погрешность. Если у взятых рядов большой коэффициент корреляции, то восстановление будет удачным, т.е. погрешность мала.

К сожалению, экспериментально не удалось определить в каком из методов наименьшая погрешность, т.к. данные в таблицы являются случайными, как и их удаление. Поэтому сравнение не было бы объективным.

В итоге мы получили восстановленную таблицу. Здесь мы воспользовались методом винзонирования. Делаем вывод, что наш метод был успешно проверен в ходе экспериментальной реализации программы на python.

```
rami@rami-K501UX:~/Desktop/itmo/System-Software/Lab2/Task-2$ python3 interface.py
Введите способ, которым хотите восстановить данные:
1. Винзорирование
2. Линеаризация
3. Корреляционное восстановление
```

Рис. 2: Пользовательский интерфейс для задания 2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	9.02059	24.6736	10.7268	15.0198	5.36386	15.5589	6.77552	15.2388	3.84847	9.69898	10.1963	26.8316	12.1492	8.7126	7.61775	
2	6.65664	26.4613	10.3625	13.7203	13.5688	27.9317	6.77762	4.37462	0	14.8387	10.4763	8.08832	3.40543	13.689	11.3228	
3	19.3647	2.67686	27.7709	10.0219	20.9879	26.9851	1.21966	27.6138	18.6359	23.0009	7.17368	14.0097	11.407	20.4712	11.5555	
4	22.685	9.8448	7.75567	25.5718	3.61969	8.77081	18.7642	26.8533	27.1154	20.6958	12.4161	0	19.7901	12.0894	10.6531	
5	9.03551	0	4.64222	0	13.2173	24.2071	27.6618	24.4949	22.9469	7.2422	4.94255	0	5.36015	25.2959	15.9097	
6	7.76383	17.5678	0	13.9947	8.77547	13.8984	10.9087	13.1192	23.3726	3.64046	8.85328	20.5656	15.3479	22.951	20.9414	
7	27.2391	22.9533	7.33514	25.907	27.609	19.9583	6.85092	5.83128	2.32235	16.5202	6.73327	12.6857	23.3822	18.6783	21.0739	
8	17.0046	25.7951	8.6897	29.1858	1.02321	22.5114	18.8317	24.8922	28.4235	1.87023	16.7368	18.7147	0	12.1541	13.9397	
9	15.0663	20.5022	27.0873	27.1733	11.1713	26.5587	6.0278	24.6002	16.5799	12.33	9.0216	1.50318	29.594	13.1223	21.7721	
10	22.5065	9.05408	17.7948	5.21919	19.8779	19.5444	5.82798	7.57679	5.16141	5.88425	10.0484	13.7197	0	9.39515	18.3509	
11	14.7779	15.1851	7.64609	9.07487	18.443	2.72504	16.7585	14.7143	2.60625	21.7921	6.81274	0	8.334	28.107	15.6732	
12	3.60265	1.128	4.19439	25.1634	26.6288	16.3145	21.7745	10.7066	27.2353	20.2815	27.0899	28.0878	19.6709	18.1281	29.2907	
13	18.7403	25.0524	4.6895	5.64745	4.55601	4.58085	13.66	22.2319	21.9213	29.6468	1.86228	7.45095	6.18266	0	20.2997	
14	5.52811	2.91889	15.353	15.9679	4.9908	1.97693	29.8188	17.7513	1.36014	21.1051	6.2034	6.4074	3.98895	25.58	17.7434	
15	29.5422	9.35081	21.861	11.4999	25.1764	24.8896	17.6645	22.1717	11.4487	11.3171	26.655	24.6721	9.79621	9.86575	8.01945	
16																

Рис. 3: Таблица до восстановления методом винзонирования

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	9.02059	24.6736	10.7268	15.0198	5.36386	15.5589	6.77552	15.2388	3.84847	9.69898	10.1963	26.8316	12.1492	8.7126	7.61775
2	6.65664	26.4613	10.3625	13.7203	13.5688	27.9317	6.77762	4.37462	4.37462	14.8387	10.4763	8.08832	3.40543	13.689	11.3228
3	19.3647	2.67686	27.7709	10.0219	20.9879	26.9851	1.21966	27.6138	18.6359	23.0009	7.17368	14.0097	11.407	20.4712	11.5555
4	22.685	9.8448	7.75567	25.5718	3.61969	8.77081	18.7642	26.8533	27.1154	20.6958	12.4161	12.4161	19.7901	12.0894	10.6531
5	9.03551	9.03551	4.64222	4.64222	13.2173	24.2071	27.6618	24.4949	22.9469	7.2422	4.94255	4.94255	5.36015	25.2959	15.9097
6	7.76383	17.5678	17.5678	13.9947	8.77547	13.8984	10.9087	13.1192	23.3726	3.64046	8.85328	20.5656	15.3479	22.951	20.9414
7	27.2391	22.9533	7.33514	25.907	27.609	19.9583	6.85092	5.83128	2.32235	16.5202	6.73327	12.6857	23.3822	18.6783	21.0739
8	17.0046	25.7951	8.6897	29.1858	1.02321	22.5114	18.8317	24.8922	28.4235	1.87023	16.7368	18.7147	18.7147	12.1541	13.9397
9	15.0663	20.5022	27.0873	27.1733	11.1713	26.5587	6.0278	24.6002	16.5799	12.33	9.0216	1.50318	29.594	13.1223	21.7721
10	22.5065	9.05408	17.7948	5.21919	19.8779	19.5444	5.82798	7.57679	5.16141	5.88425	10.0484	13.7197	13.7197	9.39515	18.3509
11	14.7779	15.1851	7.64609	9.07487	18.443	2.72504	16.7585	14.7143	2.60625	21.7921	6.81274	6.81274	8.334	28.107	15.6732
12	3.60265	1.128	4.19439	25.1634	26.6288	16.3145	21.7745	10.7066	27.2353	20.2815	27.0899	28.0878	19.6709	18.1281	29.2907
13	18.7403	25.0524	4.6895	5.64745	4.55601	4.58085	13.66	22.2319	21.9213	29.6468	1.86228	7.45095	6.18266	6.18266	20.2997
14	5.52811	2.91889	15.353	15.9679	4.9908	1.97693	29.8188	17.7513	1.36014	21.1051	6.2034	6.4074	3.98895	25.58	17.7434
15	29.5422	9.35081	21.861	11.4999	25.1764	24.8896	17.6645	22.1717	11.4487	11.3171	26.655	24.6721	9.79621	9.86575	8.01945

Рис. 4: Таблица после восстановления методом винзонирования

Анализ данных

Задание:

1) Создать таблицу nxn, заполненную случайными величинами в диапазоне от 1 до 30.

2) Реализовать алгоритм математическое ожидание и дисперсию для каждого из рядов созданной таблицы.

3) Реализовать алгоритм, определяющий наличие коррелируемых между собой рядов, если известно, что зависимость может носить линейный или экспоненциальный характер, а допустимая погрешность взаимосвязи не может превышать процент, задаваемый пользователем.

Решение:

Первая часть:

Создаем таблицу 15x15, заполненную случайными величинами в диапазоне от 1 до 30.

```
def initTable():
        for i in range(1,16):
                for j in range(1,16):
                        val = random.uniform(1,30)
                        ws.cell(column=j, row=i, value=val)
        wb.save("Data.xlsx")
```

Вторая часть:

Реализуем алгоритм математического ожидания и дисперсии для каждого из рядов созданной таблицы.

Используем представленную ниже формулу математического ожидания, где p=1/15, т.к. для всех 15 чисел одинаковая вероятность.

$$M[X] = \sum_{i=1}^{\infty} x_i \, p_i.$$

Реализуем ее на python:

```
def mean_row(i):
        summ = 0
        for j in range(1, 16):
                        summ += ws[get_column_letter(j) + str(i)].value
        return summ / 15
```

Используем представленную ниже формулу дисперсии случайной величины, где p=1/15, т.к. для всех 15 чисел одинаковая вероятность.

$$D[X] = \sum_{i=1}^{n} p_i (x_i - M[X])^2,$$

Реализуем ее на python:

```
def var(i):
        sq_summ = 0
        for j in range(1, 16):
                sq_summ += ( ws[get_column_letter(j) + str(i)].value ) ** 2

        return sq_summ / 15 - mean_row(i) ** 2
```

Третья часть:

Реализуем алгоритм, определяющий наличие коррелируемых между собой рядов, если известно, что зависимость может носить линейный или экспоненциальный характер, а допустимая погрешность взаимосвязи не может превышать процент, задаваемый пользователем.

$$r_{xy} = \frac{\sum_{i=1}^{m}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \overline{x})^2 \sum_{i=1}^{m}(y_i - \overline{y})^2}} = \frac{cov(x,y)}{\sqrt{s_x^2 s_y^2}},$$

где $\overline{x}, \overline{y}$ – выборочные средние x^m и y^m, s_x^2, s_y^2 – выборочные дисперсии, $r_{xy} \in [-1,1]$.

Коэффициент корреляции Пирсона называют также теснотой линейной связи:

- $|r_{xy}| = 1 \Rightarrow x, y$ линейно зависимы,
- $r_{xy} = 0 \Rightarrow x, y$ линейно независимы.

Формула для рассчета коэффициента корреляции подходит лишь для определения линейной зависимости между двумя рядами данных. Для определения наличия корреляции между двумя рядами данных была реализована следущая функция, которая возвращает коэффициенты линейной и экспоненциальной корелляции:

```
def corr(i, j):
        x_var = var(i)
        y_var = var(j)

        x_centered = centered_lin(i)
        y_centered = centered_lin(j)

        x_exp_centered = centered_exp(i)
        x_exp_var = var_exp(i)

        r_lin = mean_list(x_centered, y_centered)
        R_lin = r_lin / ( sqrt( x_var * y_var ) )

        r_exp = mean_list(x_exp_centered, y_centered)
        R_exp = r_exp / ( sqrt( x_exp_var * y_var ) )

        result = [R_lin, R_exp]

        return result
```

Для определения коэффициента экспоненциальной зависимости вида $y = a e^{cx}$ сперва необходимо представить в виде линейной функции. Для этого прологарифмируем обе части равенства и проведем замену:

$\ln y = \ln a + x \ln c$, $\ln a = k, \ln c = b,$ $\ln y = \hat{y}$. Теперь зависимость можно рассматривать как линейную: $\hat{y} = kx + b$. Данные преобразования производятся с помощью функции centered_exp().

После расчёта коэффициентов корреляции для всех пар рядов, пользователь вводит погрешность e (в виде числа от 0 до 1), и на экран выводятся те пары рядов, чьи коэффициенты корреляции больше чем $1 - e$.

Примеры работы программы:

```
rami@rami-K501UX:~/Desktop/itmo/System-Software/Lab2/Task-3$ python3 Lab2_Task_3.py
Ряд №1 : мат.ожидание 13.983372625260206, дисперсия 75.07758493180145
Ряд №2 : мат.ожидание 21.015098642471866, дисперсия 51.86782289470125
Ряд №3 : мат.ожидание 14.011248695864234, дисперсия 83.45885465337324
Ряд №4 : мат.ожидание 17.152402532466233, дисперсия 49.26700751191953
Ряд №5 : мат.ожидание 15.67505087756783, дисперсия 36.22158331326989
Ряд №6 : мат.ожидание 13.335557840039247, дисперсия 52.47168541815651
Ряд №7 : мат.ожидание 14.0831227603655, дисперсия 79.92753631659136
Ряд №8 : мат.ожидание 12.724134685544204, дисперсия 68.90036267420447
Ряд №9 : мат.ожидание 13.422646638245403, дисперсия 51.16141993166124
Ряд №10 : мат.ожидание 14.73582151976446, дисперсия 46.7932469449961
Ряд №11 : мат.ожидание 16.086056562814147, дисперсия 89.1577239467145
Ряд №12 : мат.ожидание 15.387552194966982, дисперсия 78.59566102765837
Ряд №13 : мат.ожидание 18.643885063424932, дисперсия 50.688319578765004
Ряд №14 : мат.ожидание 16.972166263445565, дисперсия 48.04599939110483
Ряд №15 : мат.ожидание 13.200952068114406, дисперсия 59.31685608604499


Введите допустимую погрешность взаимосвязи:
0.4
Линейно коррелируют ряды:
3 --> 4, коэф.корр = 0.638345817180241
5 --> 12, коэф.корр = -0.6035928777048336
```

Так как значения задаются случайным образом, при значении погрешности 0.4 нет ни одной пары коррелирующих рядов с экспоненциальной зависимостью. Пример части вывода при погрешности 0.94:

```
Экспоненциально коррелируют ряды:
1 --> 9, коэф.корр = -0.0633993673493845
1 --> 11, коэф.корр = -0.06122653274012171
1 --> 12, коэф.корр = 0.06517686459303872
2 --> 5, коэф.корр = -0.07520842523102718
2 --> 10, коэф.корр = -0.06229089769956927
8 --> 13, коэф.корр = -0.06084731415703907
12 --> 1, коэф.корр = 0.06014330756769916
12 --> 13, коэф.корр = 0.06116930894603207
13 --> 12, коэф.корр = 0.06910307268814754
```

Дополнительное задание:

Задание:

Объяснить, почему функция, полученная благодаря МНК, не всегда подходит для прогнозирования тренда будущего состояния параметров реальных систем.

Решение:

Классический статистический метод (метод наименьших квадратов): идеален с позиций объективности, не требует знания физических закономерностей, применим как для непрерывных, так и для дискретных процессов. Он одинаково пригоден как для стационарных, так и для нестационарных процессов. Для его успешного применения требуется достаточно большое число статистических данных на участке наблюдения, без которых метод дает существенные погрешности, и функция, описывающая процесс.

Метод относится с одинаковым доверием к информации, как в начале участка наблюдения, так и в конце его. В этом – его слабое место: невозможно предсказать результаты скачка (изменения характера процесса) не только на участке упреждения (предсказания), но и на участке наблюдения.