Министерство образования и науки Российской федерации

Федеральное государственное автономное образовательное учреждение высшего образования

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

Лабораторная работа №2 по дисциплине

“ Системное программное обеспечение”

Вариант №3

Выполнили:

студенты группы R3335

Кузьминов А. Ю.

Никитенко И. Р.

Гринжола Д. Ю.

Преподаватель:

кандидат технических наук

Капитонов А. А.

Санкт-Петербург

2018

Задание 1. Метод Наименьших Квадратов.

1. Реализовать алгоритм, заполняющий таблицу неповторяющимися координатами *x* и *y*. Для нашего варианта количество координат *n* равно 18. Диапазон значений координат вводится пользователем при запуске программы. Таблица формируется в таблицах google.

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

Пример работы реализованной программы:

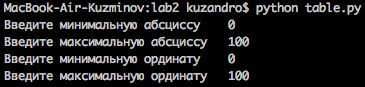


Рисунок 1. Пользовательский интерфейс.

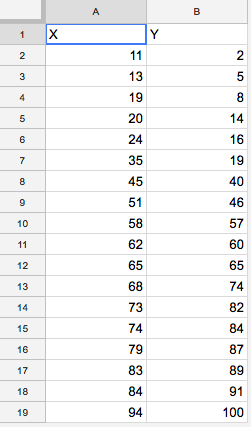


Рисунок 2. Таблица Google Sheets.

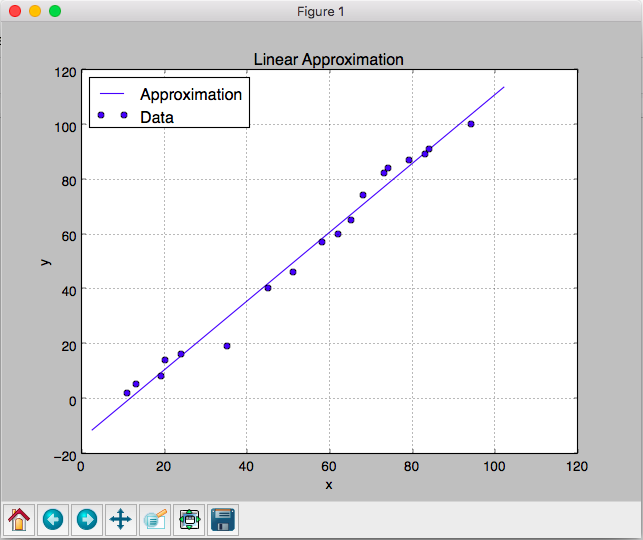


Рисунок 3. График matplotlib.

Код программы:

|  |
| --- |
| *#coding=utf8*  *#импортируем необходимые библиотеки*  **import** httplib2  **import** apiclient.discovery  **from** oauth2client.service\_account **import** ServiceAccountCredentials  **import** webbrowser  **import** random  **import** matplotlib.pyplot **as** plt  **import** numpy **as** np  **import** math    rowsNumber = 18  **def** createData():  xmin = int(raw\_input("Введите минимальную абсциссу**\t**"))  xmax = int(raw\_input("Введите максимальную абсциссу**\t**"))  **while** (xmax <= xmin):  xmax = int(raw\_input("Введите максимальную абсциссу (больше минимальной)**\t**"))  ymin = int(raw\_input("Введите минимальную ординату**\t**"))  ymax = int(raw\_input("Введите максимальную ординату**\t**"))  **while** (ymax <= ymin):  ymax = int(raw\_input("Введите максимальную ординату (больше минимальной)**\t**"))    coords = []    xValues = set([random.randint(xmin, xmax) **for** i **in** range(rowsNumber)])  **while**(len(xValues) != rowsNumber):  xValues.add(random.randint(xmin, xmax))    yValues = set([random.randint(ymin, ymax) **for** i **in** range(rowsNumber)])  **while**(len(yValues) != rowsNumber):  yValues.add(random.randint(ymin, ymax))    xValues = list(xValues)  yValues = list(yValues)  xValues.sort()  yValues.sort()  **for** i **in** range(rowsNumber):  coords.append([])  coords[i].append(xValues[i])  coords[i].append(yValues[i])    **return** coords      CREDENTIALS\_FILE = 'SysPO-64f3ebc3b4ff.json' *# имя файла с закрытым ключом*    credentials = ServiceAccountCredentials.from\_json\_keyfile\_name(CREDENTIALS\_FILE, ['https://www.googleapis.com/auth/spreadsheets',  'https://www.googleapis.com/auth/drive'])  httpAuth = credentials.authorize(httplib2.Http())  service = apiclient.discovery.build('sheets', 'v4', http = httpAuth)    spreadsheet = service.spreadsheets().create(body = {  'properties': {'title': 'Мой документ', 'locale': 'ru\_RU'},  'sheets': [{'properties': {'sheetType': 'GRID',  'sheetId': 0,  'title': 'Лист1',  'gridProperties': {'rowCount': 19, 'columnCount': 2}}}]  }).execute()      driveService = apiclient.discovery.build('drive', 'v3', http = httpAuth)  shareRes = driveService.permissions().create(  fileId = spreadsheet['spreadsheetId'],  body = {'type': 'anyone', 'role': 'reader'}, *# доступ на чтение кому угодно*  fields = 'id'  ).execute()    link = "https://docs.google.com/spreadsheets/d/" + str(spreadsheet['spreadsheetId']) + "/edit"    results = service.spreadsheets().values().batchUpdate(spreadsheetId = spreadsheet['spreadsheetId'], body = {  "valueInputOption": "USER\_ENTERED",  "data": [  {"range": "Лист1!A1:B1",  "majorDimension": "ROWS", *# сначала заполнять ряды, затем столбцы (т.е. самые внутренние списки в values - это ряды)*  "values": [["X", "Y"]]},  ]  }).execute()    coords = createData()  results = service.spreadsheets().values().batchUpdate(spreadsheetId = spreadsheet['spreadsheetId'], body = {  "valueInputOption": "USER\_ENTERED",  "data": [  {"range": "Лист1!A2:B19",  "majorDimension": "ROWS", *# сначала заполнять ряды, затем столбцы (т.е. самые внутренние списки в values - это ряды)*  "values": coords},  ]  }).execute()    webbrowser.open\_new(link)    sumx = 0.0  sumy = 0.0  sumx2 = 0.0  sumxy = 0.0    **for** i **in** range(rowsNumber):  sumx += coords[i][0]  sumx2 += coords[i][0]\*\*2  sumy += coords[i][1]  sumxy += coords[i][0]\*coords[i][1]    a = float((rowsNumber\*sumxy - sumx\*sumy)/(rowsNumber\*sumx2 - sumx\*\*2))  b = (sumy - a\*sumx)/rowsNumber    **print**("Rows Number " + str(rowsNumber))  **print**("Sum X " + str(sumx))  **print**("Sum Y " + str(sumy))  **print**("Sum XY " + str(sumxy))  **print**("Sum X2 " + str(sumx2))    length = math.fabs(coords[rowsNumber-1][0] - coords[0][0])  leftBorder = coords[0][0] - 0.1\*length  rightBorder = coords[rowsNumber-1][0] + 0.1\*length  x = np.linspace(leftBorder, rightBorder, 1000)  *#print(x)*  y = []  **for** i **in** range(len(x)):  y.append(a\*x[i]+b)  *#print(y)*  **print**(a)  **print**(b)  xDots = []  yDots = []  **for** i **in** range(rowsNumber):  xDots.append(coords[i][0])  yDots.append(coords[i][1])  plt.plot(x, y, label='Approximation')  plt.plot(xDots, yDots, 'bo',label='Data')    plt.xlabel('x')  plt.ylabel('y')    plt.title("Linear Approximation")    plt.legend(loc=2)  plt.grid()  plt.show() |

Задание 2. Восстановление данных.

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.

Пример работы реализованной программы:

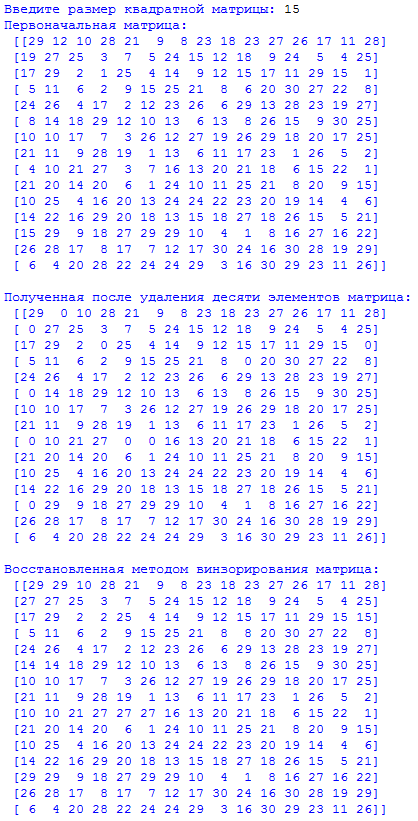


Рисунок 4. Пользовательский интерфейс ч.1.

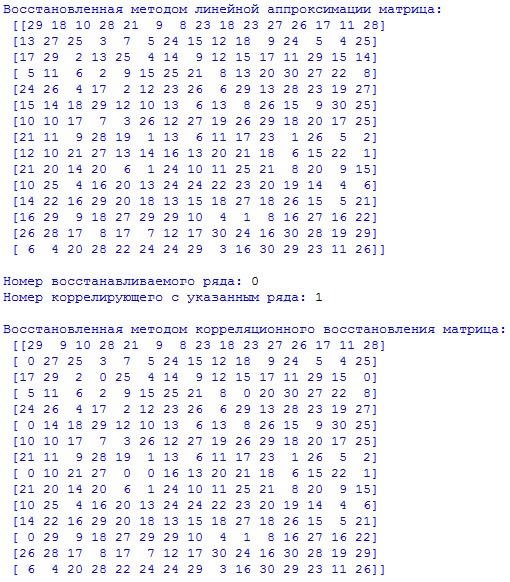


Рисунок 5. Пользовательский интерфейс ч.2.

Код программы:

|  |
| --- |
| **import** random, numpy **as** np    *#функция для создания матрицы со случайными элементами от 1 до 30*  **def** creatArray():  n = int(input('Введите размер квадратной матрицы: '))  **return** [[random.randint(1,30) **for** i **in** range(n)] **for** j **in** range(n)]      *#создаем матрицу и выводим на экран*  A = np.array(creatArray())  **print**('Первоначальная матрица: **\n**', A, '**\n**')      *#удаление десяти случайных элементов (считаем, что если элемент равен нулю, он удален)*  A\_del = np.copy(A)  zero = 0 *#количество удаленных элементов*  **while** zero != 10:  *#берем случайный элемент*  i\_rand = random.randint(0, A\_del.shape[0]-1)  j\_rand = random.randint(0, A\_del.shape[0]-1)  *#проверяем, удален ли элемент, и удаляем, если нет*  **if** A\_del[i\_rand][j\_rand] == 0:  **continue**  **else**:  A\_del[i\_rand][j\_rand] = 0  zero += 1    **print**('Полученная после удаления десяти элементов матрица: **\n**', A\_del, '**\n**')      *#метод винзорирования*  A\_vin = np.copy(A\_del)    **for** i **in** np.arange(A\_vin.shape[0]):  **for** j **in** np.arange(A\_vin.shape[1]):  **if** A\_vin[i, j] == 0:  **if** A\_vin[i, j-1] != 0 **and** (j-1) >= 0:  A\_vin[i, j] = A\_vin[i, j-1]  **else**:  **if** A\_vin[i, j+1] != 0 **and** (j+1) <= (A\_vin.shape[0] - 1):  A\_vin[i, j] = A\_vin[i, j+1]  **else**:  A\_vin[i, j] = 1    **print**('Восстановленная методом винзорирования матрица: **\n**', A\_vin, '**\n**')      *#метод линейной аппроксимации*  A\_lin = np.copy(A\_del)    **for** i **in** np.arange(A\_lin.shape[0]):  **for** j **in** np.arange(A\_lin.shape[1]):  **if** A\_lin[i, j] == 0:  A\_lin[i, j] = np.mean(A\_lin[i])    **print**('Восстановленная методом линейной аппроксимации матрица: **\n**', A\_lin, '**\n**')      *#метод корреляционного восстановления*  A\_cor = np.copy(A\_del)    n = int(input('Номер восстанавливаемого ряда: '))  m = int(input('Номер коррелирующего с указанным ряда: '))    **try**:  **for** j **in** range(0, A\_cor.shape[0]):  **if** A\_cor[n, j] == 0:  **if** j != A\_cor.shape[0]:  A\_cor[n, j] = (A\_cor[n, j+1]\*A\_cor[m, j+1])/A\_cor[m, j]  **elif** j == A\_cor.shape[0]: *#для послежнего столбца*  A\_cor[n, j] = (A\_cor[n, j-1]\*A\_cor[m, j-1])/A\_cor[m, j]  **except** BaseException: *#избегаем ошибки в случае, если утеряно слишком много данных*  **print**('**\n**Восстановить удалось не все данные')  **pass**    **print**('**\n**Восстановленная методом корреляционного восстановления матрица: **\n**', A\_cor, '**\n**') |

Анализ методов восстановления:

|  |  |  |
| --- | --- | --- |
|  | Достоинства | Недостатки |
| Винзозирование | Прост в реализации; восстановление всех утерянных данных | Большая погрешность |
| Лин. аппроксимация | Высокая точность при линейной (или близкой к таковой) зависимости данных | Реализация сложнее, чем у метода винзорирования; большая погрешность при аномальных значениях |
| Корреляция | Небольшая погрешность | Сложная реализация; есть вероятность не восстановить все данные |

Задание 3. Анализ данных.

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.

Пример работы реализованной программы:

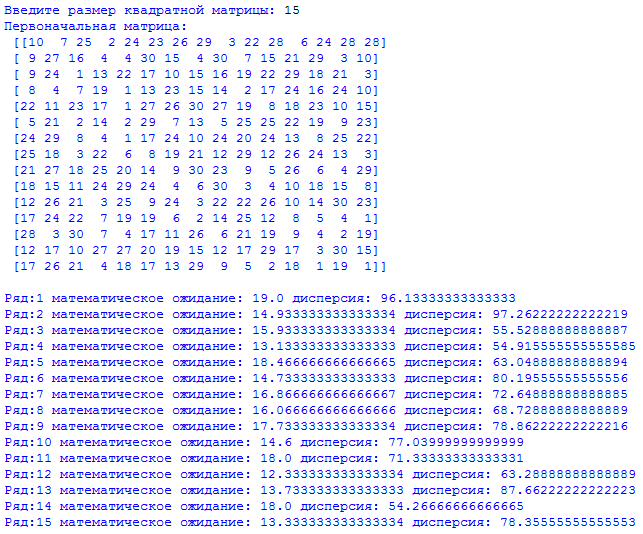


Рисунок 6. Пользовательский интерфейс ч.1.

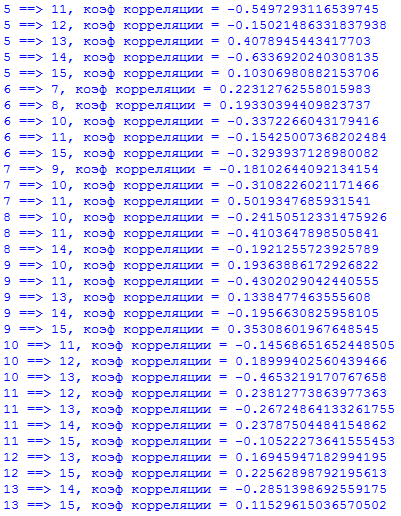
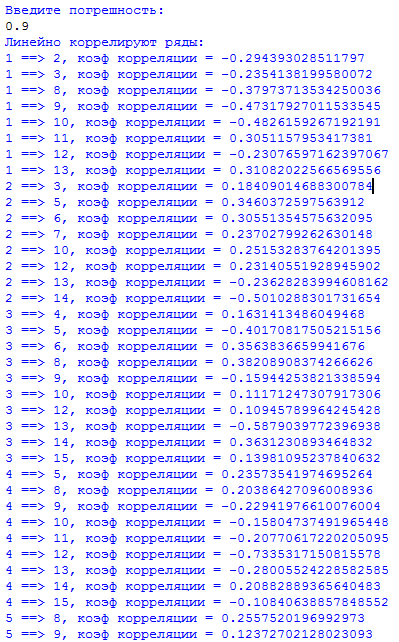


Рисунок 7. Пользовательский интерфейс ч.2.

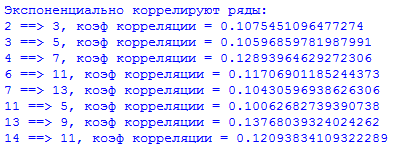


Рисунок 8. Пользовательский интерфейс ч.3.

Код программы:

|  |
| --- |
| *#coding=utf8*  **import** random, numpy **as** np  **from** math **import** sqrt, log    *#функция для создания матрицы со случайными элементами от 1 до 30*  n = int(input('Введите размер квадратной матрицы: '))  **def** creatArray(n):  **return** [[random.randint(1,30) **for** i **in** range(n)] **for** j **in** range(n)]      *#создаем матрицу и выводим на экран*  A = np.array(creatArray(n))  **print**('Первоначальная матрица: **\n**', A, '**\n**')    mathExpVals = []  despVals = []    **def** rowMathExp(i):  sigma = 0.0  **for** j **in** range(n):  sigma += A[i][j]  mathExp = sigma/15  **return** mathExp    **def** rowDispersion(i):  sigma = 0  **for** j **in** range(n):  sigma += A[i][j]\*\*2  dispersion = (sigma/n) - rowMathExp(i)\*\*2  **return** dispersion    **def** mathExpAndDisp():  **for** i **in** range(n):  mathExpVals.append(rowMathExp(i))  despVals.append(rowDispersion(i))    **def** mathExpLists(x,y):  sigma = 0  **for** i **in** range(n):  sigma += x[i]\*y[i]  **return** sigma/n    **def** mathExpList(x):  sigma = 0  **for** a **in** x:  sigma += a  **return** sigma/n    **def** centeredLin(i):  x = []  mathExpX = rowMathExp(i)  **for** j **in** range(n):  x.append(A[i][j] - mathExpX)  **return** x      **def** centeredExp(i):  x = []  **for** j **in** range(n):  x.append(log(A[i][j]))  mathExpX = mathExpList(x)  **for** j **in** range(n):  x[j] = x[j] - mathExpX  **return** x    **def** dispExp(i):  x = []  **for** j **in** range(n):  x.append(log(A[i][j]))  xMathExp = mathExpList(x)  **for** j **in** range(n):  x[j] = x[j] \*\* 2  sigma = 0  **for** j **in** range(n):  sigma += x[j] \*\* 2  disp = sigma / n - xMathExp \*\* 2  **return** disp    **def** dispersion(i):  sigma = 0  **for** j **in** range(n):  sigma += A[i][j]\*\*2  **return** sigma/n - rowMathExp(i)\*\*2    **def** cor(i,j):  xVar = dispersion(i)  yVar = dispersion(j)  xCentered = centeredLin(i)  yCentered = centeredLin(j)  xExpCentered = centeredExp(i)  xExpDisp = dispExp(i)  rLin = mathExpLists(xCentered, yCentered)  RLin = rLin / (sqrt(abs(xVar \* yVar)))  rExp = mathExpLists(xExpCentered, yCentered)  RExp = rExp / (sqrt(abs(xExpDisp \* yVar)))  result = [RLin, RExp]  **return** result      mathExpAndDisp()  **for** i **in** range(n):  **print**("Ряд:" + str(i+1) + " математическое ожидание: " + str(mathExpVals[i]) + " дисперсия: " + str(despVals[i]))    corCoef = []    **for** i **in** range(n):  **for** j **in** range(n):  corCoef.append(cor(i, j))    **print**("**\n\n**Введите погрешность: ")  error = float( input() )    **print**("Линейно коррелируют ряды: ")  **for** i **in** range(n):  **for** j **in** range(i+1, n):  **if** abs(corCoef[i \* n + j][0]) > 1 - error:  **print**(str(i + 1) + " ==> " + str(j + 1) + ", коэф корреляции = " + str(corCoef[i \* n + j][0]))      **print**("**\n\n**Экспоненциально коррелируют ряды: ")  **for** i **in** range(n):  **for** j **in** range(n):  **if** i == j:  **continue**  **if** (abs(corCoef[i \* (n-1) + j][1]) > 1 - error):  **print**(str(i + 1) + " ==> " + str(j + 1) + ", коэф корреляции = " + str(corCoef[i \* (n-1) + j][1])) |

Дополнительное задание.

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

Решение:

Метод наименьших квадратов (МНК) — математический метод, применяемый для решения различных задач, основанный на минимизации суммы квадратов отклонений некоторых функций от искомых переменных. Он может использоваться для «решения» неопределенных систем уравнений (когда количество уравнений превышает количество неизвестных), для поиска решения в случае обычных (не переопределенных) нелинейных систем уравнений, для аппроксимации точечных значений некоторой функции.

Функция, полученная благодаря МНК не всегда подходит для прогнозирования тренда будущего состояния параметров реальных систем, так как с помощью данного метода невозможно предсказать резкие изменения данных (скачки), что не редко случается на практике как на участке наблюдения, так и на участке предсказания.

Ссылка на GitHub: <https://github.com/nikotheflow/Python-labs>