Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Кафедра Систем Управления и Информатики

Экзаменационная работа:
Написание прикладной программы на языке Python

Выполнил: Антипов В.А.

Проверили: Мусаев А.А.

Санкт-Петербург
2018

Задача

Необходимо написать программу для нахождения функциональной зависимости параметров. В качестве параметров дана таблица экспериментальных измерений в GoogleSheets. Зависимость параметров ищется по трем заданным функциям:

$$y = A \cdot exp(t), \quad y = A \cdot t^2, \quad y = A \cdot t + B$$

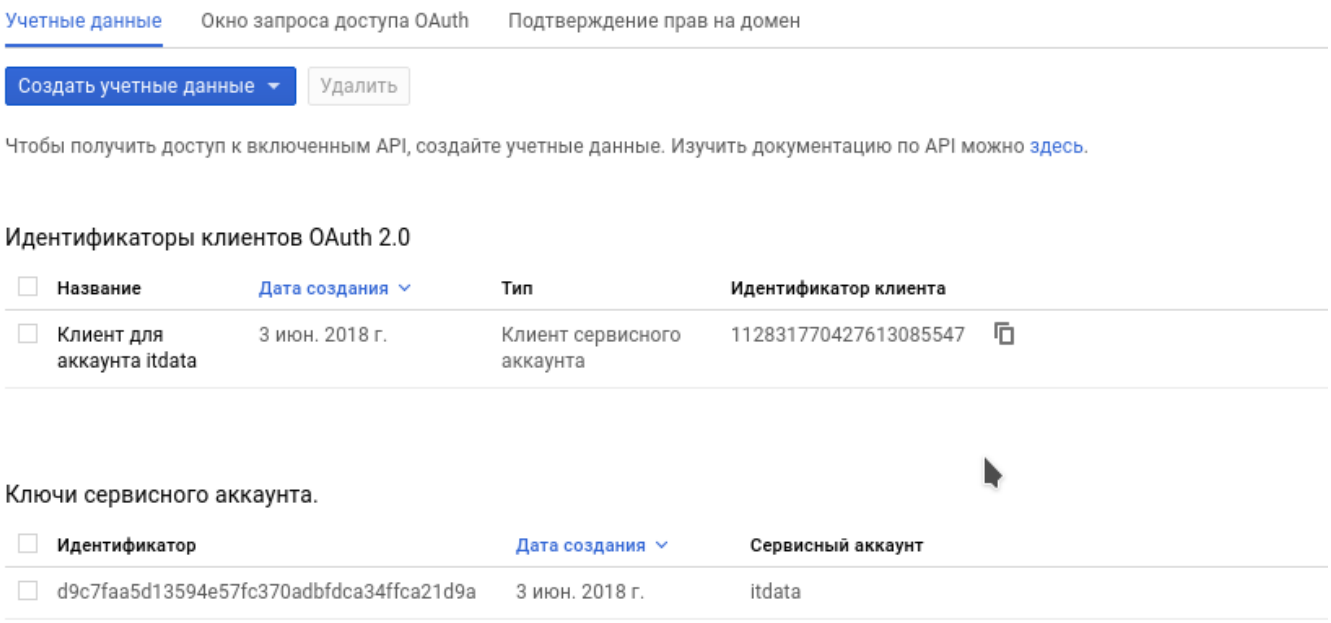
Также программа должна учитывать введенное пользователем максимальное относительное отклонение связи. Отклонение задается пользователем в процентах. Результат выполнения программы должен сохраняться в текстовый файл в виде:
"Параметр 1 связан с Параметром 2, зависимость линейная".

Выполнение

Настройка аккаунта и установка пакетов

Для обращения к Google таблицам необходимо создать учетную запись и получить доступ к GoogleSheets API и GoogleDrive API.

Для этого необходимо:
- Создать Google Аккаунт, если он отсутствует.
- Войти в *Google API Console*
- Создать новый проект и подключить к нему GoogleSheets API и GoogleDrive API.
- Сконфигурировать JSON ключ данной учетной записи для подключения к GoogleSheets через OAuth 2.
- Добавить аккаунт "робота"в группу разработчиков проекта (таблицы).



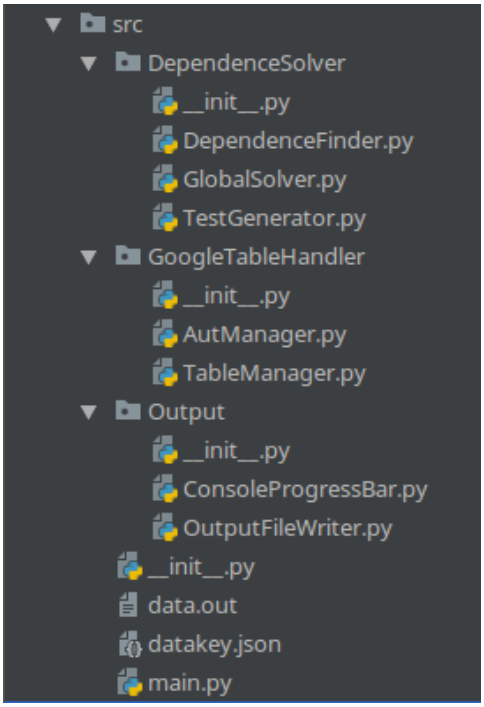
Созданные сервисный аккаунт и полученный json ключ

Теперь перейдем к установке пакетов Python. Воспользуемся удобным пакетным менеджером Pip3 для установки пакетов gspread и OAuth2Client, необходимых для работы с google таблицами и аутентификации в сервисах google.

```
pip3 install gspread oauth2client
```

Написание программы

Программа разделена на несколько частей, сформировавших пакеты:
1) Аутентификация в GoogleSheets и парсинг данных из таблицы.(*GoogleTableHandler*
2) Установка и поиск функциональной зависимости.(*DependenceSolver*)
3) Вывод результатов и прогресса решения в консоль.(*Output*)



Структура проекта, иерархия пакетов

1) Аутентификация в GoogleSheets и парсинг данных из таблицы.(*GoogleTableHandler*)

Модуль AuthManager предназначен для аутентификации в сервисах google посредством протокола OAuth2. OAuth2 - это сравнительно новый протокол для авторизации, основанный на http-запросах. С помощью него можно выдавать права другим программам, таким как наша, на редактирование данных от лица пользователя без аутентификации под новым пользователем.

Модуль *AuthManager*:

```
def authorized():
    while True:
        autkey = input("Введите путьдоключааутентификации    : ")
        scope = ['https://spreadsheets.google.com/feeds', 'https://www.googleapis.com
            /auth/drive']
        try:
            creds = ServiceAccountCredentials.from_json_keyfile_name(autkey, scope)
            client = gspread.authorize(creds)
            print("[Google Sheets] Authentification Complete!")
            return client
        except FileNotFoundError:
            print("Файл "+ autkey + " ненайден !")
        except json.decoder.JSONDecodeError:
            print("Файл неявляетсяключом  !")
            continue
```

Для подключения используется json-ключ, хранящий в себе данные(ключ) для доступа к возможностям аккаунта без аутентификации (логина и пароля).
После запроса ввести путь до json-ключа в программе записываются в лист сервисы для которых мы хотим получить *authorization code* (код полученный для приложения от сервера, необходимый для последующего формирования запросов). Далее происходит формирование полномочий и авторизация:

```
ServiceAccountCredentials.from_json_keyfile_name(autkey, scope)
client = gspread.authorize(creds)
```

Если будет поймана ошибка о неверности json ключа или его отсутствии - пользователь будет вынужден ввести путь до него еще раз.
После успешной авторизации метод *def authorized()* вернен обьект типа *client*, с помощью него

и будет происходить получение и запись данных в таблицу.

Обработкой данных из таблицы занимается класс *TableManager*. Класс формирует удобные для обработки листы с параметрами из полученной с помощью *client*-а таблицы.
Метод класса *def __openTable(self)* открывает таблицу по имени или url с помощью *client*-а и в случае возникновения ошибки - говорит об этом.

```
def __openTable(self):
    try:
        if re.match("http", self.__tablename) :
            table=self.__client.open_by_url(self.__tablename).sheet1
        else:
            table = self.__client.open(self.__tablename).sheet1
    except gspread.exceptions.SpreadsheetNotFound:
        print("[ERROR]: You can not open this GoogleSheets!")
        exit(5)
    return table
```

Метод класса *def getParamList(self)* формирует список параметров в листе проверяя корректность полей, отсутствие данных в ячейке и переводя все числа в тип *float128* для работы с длинной арифметикой с помощью NumPy.

```
def getParamList(self):
    tableList= self.__getTableListfromEdit()
    paramList = []
    countMetrics = 0
    for parameter in tableList:
        if re.match("Parameter", parameter[0]):
            if (countMetrics != 0) and (len(parameter[1:]) != countMetrics):
                print("Count parameter error!")
                exit(6)
            paramList.append(list(map(float128, parameter[1:])))
            countMetrics = len(paramList[0])
    return paramList
```

Таблица принимается в следующем виде:

	A	B	C	D
1	Parameter 1	1	2	3
2	Parameter 2	-6.85	-18.62	-50.616
3	Parameter 3	-0.001	0	0
4	Parameter 4	2.664	7.241	19.684
5	Parameter 5	-4.997	-10.489	-25.421
6	Parameter 6	2.08	2.08	2.08
7	Parameter 7	2.574	2.574	2.574
8	Parameter 8	7.136	20.024	55.06
9	Parameter 9	-5.97	-5.97	-5.97
10	Parameter 10	-1.389	-1.39	-1.39
11	Parameter 11	7.639	23.685	67.305
12	Parameter 12	-0.002	0	0
13	Parameter 13	-74.607	-551.26	-4073.547
14	Parameter 14	-0.001	0	0
15	Parameter 15	1413.442	559068763.8	9.19193E+23
16	Parameter 16	0	0	0
17	Parameter 17	1.42	1.42	1.42
18	Parameter 18	-2387.146	-944205023.2	-1.55241E+24
19	Parameter 19	0	0	0
20	Parameter 20	176.92	1292.288	9534.44
21				

Тестовая сгенерированная таблица параметров.

Метод *def setParamList(self, list)* позволяет записывать построчно в таблицу данные полученные в качестве листа, как параметра вызова. Данный метод будет необходим для записи в таблицу тестовых случайно сгенерированных данных для проверки работы программы.

```
def setParamList(self, list):
    self.__table.clear()
    for i in range(len(list)):
        self.__table.delete_row(i+1)
        self.__table.insert_row(([" Parameter " + str(i+1)] + list[i]), i+1)
```

2) Установка и поиск функциональной зависимости.(*DependenceSolver*)

Основой данного пакета и всей программы в целом является класс *DependenceFinder*. В данном классе реализованы методы для поиска функциональной зависимости параметров.
Алгоритм поиска зависимостей основан на МНК - методе наименьших квадратов. Метод наименьших квадратов наиболее прост в программной реализации. Также был вариант использовать метод дифференцирования, но он был отсечен, ввиду малого количества экспериментальных точек, что критично для численного дифференцирования. Сам метод заключается в поиске неопределенных коэффициентов уравнения, так чтобы сумма квадратичных ошибок отклонения экспериментальных данных от этого уравнения была минимальной, т.е:

$$e = y - y^* \quad S = \sum_{i=1}^{n} e^2$$

, где: e - отклонение, y - вид функции по которой будет происходить аппроксимация, S - сумма квадратов ошибок. Принцип МНК именно в нахождении таких коэффициентов у функции y, чтобы $S \to 0$.

Выведенные формулы для аппроксимации данных нам функций:

1) Линейная аппроксимация ($y = A \cdot x + B$):

$$A = \frac{n \cdot \sum_{i=1}^{n}(y_i^* \cdot x_i) - \sum_{i=1}^{n} y_i^* \cdot \sum_{i=1}^{n} x_i}{n \cdot \sum_{i=1}^{n} x_i^2 + (\sum_{n=1}^{n} x_i)^2} \quad B = \frac{\sum_{i=1}^{n} y_i^* - A \cdot \sum_{i=1}^{n} x_i}{n}$$

2) Экспоненциальная аппроксимация ($y = A \cdot e^x$):

$$A = exp\Big(\frac{\sum_{i=1}^{n}(ln(y_i^*) + x_i)}{n}\Big)$$

3) Квадратичная аппроксимация ($y = A \cdot x^2$)

$$A = \frac{\sum_{i=1}^{n} x_i^2 \cdot y_i}{\sum_{i=1}^{n} x_i^4}$$

```
def expAproxMNK(self, param1, param2):
    if not self.__checkAriphmetics(1e-10, 1e+20, 1e-10, 1e+5, param1, param2, True):
        return [[0.0], 0.0, False]
    return self.__getAexp(param1, param2)

def linearAproxMNK(self, param1, param2):
    if not self.__checkAriphmetics(1e-10, 1e+20, 1e-10, 1e+20, param1, param2, False)
        :
        return [[0.0, 0.0], 0.0, False]
    return self.__getABlin(param1, param2)


def quadAproxMNK(self, param1, param2):
    if not self.__checkAriphmetics(1e-10, 1e+20, 1e-10, 1e+10, param1, param2, False)
        :
        return [[0.0], 0.0, False]
    return self.__getAquad(param1, param2)
```

После поиска коэффициентов, происходит подсчет максимального отклонения экспериментальных данных от значения аппроксимирующей функции и сравнение этого отклонения с введенным пользователем максимальным отклонением.

Метод класса *def __checkAriphmetics(self, minparam1, maxparam1, minparam2, maxparam2, param1, param2, checkNegativeNumber)* проверяет принадлежат ли параметры интервалу допустимых числовых значений, это было введенно для отсечения вычислений приводящих к переполнению памяти, выделенной на переменную. Также подобная проверка отсекает поиск функциональной зависимости при неизменных параметрах. В случае обнаружения недопустимо малых, повторяющихся или недопустимо больших чисел метод возращает False, иначе True.

```
def __checkAriphmetics(self, minparam1, maxparam1, minparam2, maxparam2, param1,
    param2, checkNegativeNumber):
    n = len(param1)
    countzero1 = 0
    countzero2 = 0
    repeat = param2[0]
    countrepeat = 0

    for i in range(n):
        if i != 0 and param2[i] == repeat:
            countrepeat += 1
        repeat = param2[i]
        if abs(param1[i]) < minparam1 or abs(param2[i]) > maxparam1 or (
            checkNegativeNumber and param1[i] < 0.0):
```

```
            countzero1 += 1
        if abs(param2[i]) < minparam2 or abs(param2[i]) > maxparam2:
            countzero2 += 1
    if countzero1 != 0 or countzero2 != 0 or countrepeat != 0:
        return False
    else: return  True
```

В следующих методах происходит сама аппроксимация, при этом на выходе мы получаем лист, который хранит в себе результат аппроксимации, то есть: коэффициенты, максимальное найденное отклонение и флаг о прохождении проверки. Для аппроксимации было решено использовать lambda выражения для сокращения записи и удобочитаемости.

Квадратичная аппроксимация($y = A \cdot x^2$):

```
def __getAquad(self, param1, param2):
    maxRelError = -1
    n = len(param1)
    a = np.sum(list(map(lambda x, y: x * x * y, param2, param1))) / np.sum(list(map(
        lambda x: x ** 4, param2)))

    if abs(a) < 1e-10: return [[0], 0.0, False]
    for i in range(n):
        relError = -1
        if param2[i] != 0:
            relError = np.abs((a * param2[i] ** 2 - param1[i]) / (a * param2[i] ** 2)
                )

        if relError > maxRelError:
            maxRelError = relError
    if maxRelError == -1 or maxRelError > self.__maxRelativeError:
        return [[0], 0.0, False]
    else:
        return [[a], maxRelError, True]
```

Линейная аппроксимация($y = A \cdot x + B$):

```
def __getABlin(self, param1, param2):
    maxRelError = -1
    n=len(param1)

    a = (n * (np.sum(list(map(lambda x, y: x * y, param2, param1)))) - np.sum(param1)
        * np.sum(param2)) / (
                n * np.sum(list(map(lambda x: x * x, param2))) - np.sum(param2) ** 2)
    b = (np.sum(param1) - a * np.sum(param2)) / n

    if a == 0.0: return [[a], 0.0, False]
    for i in range(n):
        if param2[i] != 0.0:
            relError = np.abs(((a * param2[i] + b) - param1[i]) / (a * param2[i] + b)
                )
        else:
            relError = np.abs(param1[i])

        if relError > maxRelError:
            maxRelError = relError

    if maxRelError == -1 or maxRelError > self.__maxRelativeError:
        return [[0, 0], 0.0, False]
    else:
        return [[a, b], maxRelError, True]
```

Экспоненциальная аппроксимация($y = A \cdot e^x$):

```
def __getAexp(self, param1, param2):
    maxRelError = -1
    n = len(param1)
    a = np.exp(np.sum(list(map(lambda x, y: (np.log(y) - x), param2, param1))) / n)

    for i in range(n):
        relError = np.abs((a * np.exp(param2[i]) - param1[i]) / (a * np.exp(param2[i
            ])))
        if relError > maxRelError:
            maxRelError = relError
    if maxRelError == -1 or maxRelError > self.__maxRelativeError:
        return [[0], 0.0, False]
    else:
        return [[a], maxRelError, True]
```

2.1) Генератор зависимостей для тестирования:

Следующий класс данного пакета *TestGenerator*. В нем происходит непосредственно генерация тестовых данных для тестирования искателя зависимостей параметров. Данный класс был написан исключительно для тестирования, в нем используется random позволяющий создать случайное отклонение от желаемой функциональной зависимости. Параметры для генерации зависимости а также функция также выбираются случайно.

```
def __expGen(self):
    a = random.randint(-self.__maxvalue*4, self.__maxvalue*4)/2.0
    error = random.randint(-self.__randomError, self.__randomError)/100.0
    numberparam1 = random.randint(0, len(self.__currentGenParameters)-1)
    param1 = np.array(self.__currentGenParameters[numberparam1], dtype=np.float128)
    param2 = list(map(lambda x: float(np.round(((a*np.exp(x)).real + (a*np.exp(x)*
        error).real), 3)), param1))
    return param2

def __linGen(self):
    a = random.randint(-self.__maxvalue*4, self.__maxvalue * 4) / 2.0
    b = random.randint(-self.__maxvalue*4, self.__maxvalue * 4) / 2.0
    error = random.randint(-self.__randomError, self.__randomError) / 100.0
    numberparam1 = random.randint(0, len(self.__currentGenParameters) - 1)
    param1 = np.array(self.__currentGenParameters[numberparam1], dtype=np.float128)
    param2 = list(map(lambda x: float(np.round(((a * x + b) + (a * x +b) * error), 3)
        ), param1))
    return param2


def __quadGen(self):
    a = random.randint(-self.__maxvalue*4, self.__maxvalue * 4) / 2.0
    error = random.randint(-self.__randomError, self.__randomError) / 100.0
    numberparam1 = random.randint(0, len(self.__currentGenParameters) - 1)
    param1 = np.array(self.__currentGenParameters[numberparam1], dtype=np.float128)
    param2 = list(map(lambda x: float(np.round((a*x*x + a*x*x*error), 3)), param1))
    return param2
```

Класс *GlobalSolver()* является оберткой для всех вышеперечисленных классов, он является основой программы, в нем создаются обьекты DependenceFinder и TestGenerator, а также определяются их параметры.

В функции класса *def __open(self)* происходит:
1) запрос ввода имени таблицы.
2) обработка TableManager-ом таблицы.

3) создание генератора тестовой таблицы и непосредственно сама генерация.(для конечной программы данная часть кода отсутствует).
4) Получение параметров из таблицы.
5) Запрос ввода имени для сохранения файла с результатами.
6) Запрос ввода макимально допустимого отклонения от найденной связи.
7) Создание обьекта решателя.

```
def __open(self):
    self.__client = authorized()
    self.__tablename = input("Введите названиегуглтаблицынавашемаккаунтеили url
        адресдоступнойтаблицы : ")
    self.__tableMan = TableManager(self.__client, self.__tablename)
    #self.__gen = RandomDependGenerator(20, 1, 50)
    #self.__tableMan.setParamList(self.__gen.randomGeneratedDepend())
    self.__tableMan.getParamList()
    self.__parameters = self.__tableMan.getParamList()
    self.__outfilename = input("Введите имяфайладлясохраненияезультатов : ")
    self.__fileWriter = OutputFileWriter(self.__outfilename)
    self.__maxRelError = float(input("Введите
        максимальноеотносительноеотклонениевпроцентахотзависимости : "))/100.0
    self.__finder = DependenceFinder(self.__maxRelError)
```

В методе *def solv(self)* происходит нахождение зависимостей путем перебора всех параметров, т.е. программа выполняет $n \cdot (n-1)$ итераций. Из найденных возможных связей выбирается та, что имеет наименьшее отклонение. Также здесь проинициализирован прогресс бар, для удобного мониторинга процесса расчета зависимостей. Обновление прогресс бара происходит каждый такт подбора связи между двумя параметрами.

```
def solv(self):
    progressBar = ConsoleProgressBar(50, self.__countParameters*(self.
        __countParameters-1))
    progressBar.startProcess()
    countdep=0
    for p1 in range(len(self.__parameters)):
        for p2 in range(len(self.__parameters)):
            if p1 != p2:
                #print(str(p1)+" "+str(p2))
                dep = 3
                minError = 1000000
                resultL = self.__finder.linearAproxMNK(self.__parameters[p1], self.
                    __parameters[p2])
                resultE = self.__finder.expAproxMNK(self.__parameters[p1], self.
                    __parameters[p2])
                resultQ = self.__finder.quadAproxMNK(self.__parameters[p1], self.
                    __parameters[p2])
                rel=[resultL[2], resultE[2], resultQ[2]]
                commonResultError = [resultL[1], resultE[1], resultQ[1]]

                for i in range(len(commonResultError)):
                    if commonResultError[i] < minError and commonResultError[i] <
                        self.__maxRelError and rel[i]:
                        minError = commonResultError[i]
                        dep = i
                if dep != 3:
                    self.__fileWriter.writeResult(p1, p2, self.__accesDependList[dep
                        ])
                    countdep+=1
            progressBar.updateProgress(countdep)
```

3) Вывод результатов и прогресса решения в консоль.(*Output*)

Последняя часть программы необходима для вывода результатов расчетов на экран. В этом пакете лежит класс *OutputFileWriter*, необходимый для форматированного вывода результатов в файл. Его метод *def writeResult* в каждой новой строке пишет то, какой параметр с каким связан и какой функциональной зависимостью.

```
Параметр 9 связан с параметром 10: Зависимость Линейная
Параметр 9 связан с параметром 12: Зависимость Линейная
Параметр 9 связан с параметром 19: Зависимость Линейная
Параметр 10 связан с параметром 1: Зависимость Линейная
Параметр 10 связан с параметром 3: Зависимость Линейная
Параметр 10 связан с параметром 4: Зависимость Линейная
Параметр 10 связан с параметром 7: Зависимость Линейная
Параметр 12 связан с параметром 1: Зависимость Квадратичная
Параметр 12 связан с параметром 3: Зависимость Квадратичная
Параметр 12 связан с параметром 19: Зависимость Линейная
Параметр 14 связан с параметром 7: Зависимость Экспоненциальная
Параметр 16 связан с параметром 1: Зависимость Линейная
Параметр 16 связан с параметром 3: Зависимость Линейная
Параметр 16 связан с параметром 10: Зависимость Линейная
Параметр 16 связан с параметром 12: Зависимость Линейная
Параметр 19 связан с параметром 12: Зависимость Линейная
```

Вывод результата поиска зависимостей в файл.

```
def writeResult(self, NumberParam1, NumberParam2, type): self.__file.write("Параметр "
    + str(NumberParam1) + " связанспараметром  " + str(NumberParam2) +": Зависимость "+
   type + "\n")
```

Класс *ConsoleProgressBar* был реализован для красивого, динамически обновляемого вывода текущего процесса расчета в консоль со шкалой заполнения. Его основой является метод *def updateProgress(self, countDepend)*, в нем происходит расчет необходимого количества символов, для заполнения шкалы пропорционально рашению. Динамическое обновление данных происходит благодаря использованию потокового вывода *stdout*.

```
/usr/bin/python3.5 /media/files/ITLabFinal/src/main.py
Введите путь до ключа аутентификации: datakey.json
[Google Sheets] Authentification Complete!
Введите название гугл таблицы на вашем аккаунте или url адрес доступной таблицы: dataIT
Введите имя файла для сохранения результатов: data.out
Введите максимальное относительное отклонение в процентах от зависимости: 1
[DependFinder] Solver Progress:
[🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱🐱] 100.0% ... Найдено зависимостей: 49
Process finished with exit code 0
```

Результат работы программы, вывод прогресса.

```
def updateProgress(self, countDepend):
    self.__currentProgress += 1
    self.__currentSizeBar = round(1.0*self.__currentProgress * self.__sizeIterable)
    self.__bar = u'#' * self.__currentSizeBar + ' ' * (self.__sizeLine - self.
        __currentSizeBar)

    percents = round(100.0 * self.__currentProgress / float(self.__countIteration),
        1)
    sys.stdout.write('[%s] %s%s ...%s\r' % (self.__bar, percents, '%', "
        Найденозависимостей : "+str(countDepend)+" из "+ str(self.__currentProgress) + "
        проверенных. "))
    sys.stdout.flush()
```

Приложение

Весь проект можно увидеть на GitHub:
`https://github.com/TexnoMann/DependenceFinder.git`

Пакет DependenceSolver:

Код *DependenceSolver*

```
from cmath import exp, log
import numpy as np



class DependenceFinder:

    def __init__(self, __maxRelativeError):
        self.__maxRelativeError = __maxRelativeError



    def expAproxMNK(self, param1, param2):
        if not self.__checkAriphmetics(1e-10, 1e+20, 1e-10, 1e+5, param1, param2,
            True):
            return [[0.0], 0.0, False]
        return self.__getAexp(param1, param2)

    def linearAproxMNK(self, param1, param2):
        if not self.__checkAriphmetics(1e-10, 1e+20, 1e-10, 1e+20, param1, param2,
            False):
            return [[0.0, 0.0], 0.0, False]
        return self.__getABlin(param1, param2)


    def quadAproxMNK(self, param1, param2):
        if not self.__checkAriphmetics(1e-10, 1e+20, 1e-10, 1e+10, param1, param2,
            False):
            return [[0.0], 0.0, False]
        return self.__getAquad(param1, param2)

    def __getAquad(self, param1, param2):
        maxRelError = -1
        n = len(param1)
        a = np.sum(list(map(lambda x, y: x * x * y, param2, param1))) / np.sum(list(
            map(lambda x: x ** 4, param2)))

        if abs(a) < 1e-10: return [[0], 0.0, False]
        for i in range(n):
            relError = -1
            if param2[i] != 0:
                relError = np.abs((a * param2[i] ** 2 - param1[i]) / (a * param2[i]
                    ** 2))

            if relError > maxRelError:
                maxRelError = relError
        if maxRelError == -1 or maxRelError > self.__maxRelativeError:
            return [[0], 0.0, False]
        else:
            return [[a], maxRelError, True]

    def __getABlin(self, param1, param2):
        maxRelError = -1
        n=len(param1)
```

```
50
51          a = (n * (np.sum(list(map(lambda x, y: x * y, param2, param1)))) - np.sum(
                param1) * np.sum(param2)) / (
52                      n * np.sum(list(map(lambda x: x * x, param2))) - np.sum(param2)
                           ** 2)
53          b = (np.sum(param1) - a * np.sum(param2)) / n
54
55          if a == 0.0: return [[a], 0.0, False]
56          for i in range(n):
57              if param2[i] != 0.0:
58                  relError = np.abs(((a * param2[i] + b) - param1[i]) / (a * param2[i]
                        + b))
59              else:
60                  relError = np.abs(param1[i])
61
62              if relError > maxRelError:
63                  maxRelError = relError
64
65          if maxRelError == -1 or maxRelError > self.__maxRelativeError:
66              return [[0, 0], 0.0, False]
67          else:
68              return [[a, b], maxRelError, True]
69
70      def __getAexp(self, param1, param2):
71          maxRelError = -1
72          n = len(param1)
73          a = np.exp(np.sum(list(map(lambda x, y: (np.log(y) - x), param2, param1))) /
                n)
74
75          for i in range(n):
76              relError = np.abs((a * np.exp(param2[i]) - param1[i]) / (a * np.exp(
                    param2[i])))
77              if relError > maxRelError:
78                  maxRelError = relError
79          if maxRelError == -1 or maxRelError > self.__maxRelativeError:
80              return [[0], 0.0, False]
81          else:
82              return [[a], maxRelError, True]
83
84      def __checkAriphmetics(self, minparam1, maxparam1, minparam2, maxparam2, param1,
            param2, checkNegativeNumber):
85          n = len(param1)
86          countzero1 = 0
87          countzero2 = 0
88          repeat = param2[0]
89          countrepeat = 0
90
91          for i in range(n):
92              if i != 0 and param2[i] == repeat:
93                  countrepeat += 1
94              repeat = param2[i]
95              if abs(param1[i]) < minparam1 or abs(param2[i]) > maxparam1 or (
                    checkNegativeNumber and param1[i] < 0.0):
96                  countzero1 += 1
97              if abs(param2[i]) < minparam2 or abs(param2[i]) > maxparam2:
98                  countzero2 += 1
99          if countzero1 != 0 or countzero2 != 0 or countrepeat != 0:
100             return False
101         else: return  True
```

Код *GlobalSolver*

```
1 import time
2 from click._unicodefun import click
```

```
3
4  from src.DependenceSolver.TestGenerator import RandomDependGenerator
5  from src.GoogleTableHandler.AutManager import *
6  from src.GoogleTableHandler.TableManager import *
7  from src.DependenceSolver.DependenceFinder import *
8  from src.Output.ConsoleProgressBar import *
9  from src.Output.OutputFileWriter import *
10
11
12 class GlobalSolver():
13
14     def __init__(self):
15         self.__accesDependList = ["Линейная", "Экспоненциальная", "Квадратичная", "
               Отсутствует"]
16         self.__open()
17         self.__countParameters = len(self.__parameters)
18
19     def __open(self):
20         self.__client = authorized()
21         self.__tablename = input("Введите названиегуглтаблицынавашемаккаунтеили          url
               адресдоступнойтаблицы  : ")
22         self.__tableMan = TableManager(self.__client, self.__tablename)
23         #self.__gen = RandomDependGenerator(20, 1, 50)
24         #self.__tableMan.setParamList(self.__gen.randomGeneratedDepend())
25         self.__tableMan.getParamList()
26         self.__parameters = self.__tableMan.getParamList()
27         self.__outfilename = input("Введите имяфайладлясохранениярезультатов     : ")
28         self.__fileWriter = OutputFileWriter(self.__outfilename)
29         self.__maxRelError = float(input("Введите
               максимальноеотносительноеотклонениевпроцентахотзависимости       : "))/100.0
30         self.__finder = DependenceFinder(self.__maxRelError)
31
32     def solv(self):
33         progressBar = ConsoleProgressBar(50, self.__countParameters*(self.
               __countParameters-1))
34         progressBar.startProcess()
35         countdep=0
36
37         for p1 in range(len(self.__parameters)):
38             for p2 in range(len(self.__parameters)):
39                 if p1 != p2:
40                     #print(str(p1)+" "+str(p2))
41                     dep = 3
42                     minError = 1000000
43                     resultL = self.__finder.linearAproxMNK(self.__parameters[p1],
                           self.__parameters[p2])
44                     resultE = self.__finder.expAproxMNK(self.__parameters[p1], self.
                           __parameters[p2])
45                     resultQ = self.__finder.quadAproxMNK(self.__parameters[p1], self.
                           __parameters[p2])
46                     rel=[resultL[2], resultE[2], resultQ[2]]
47                     commonResultError = [resultL[1], resultE[1], resultQ[1]]
48
49
50                     for i in range(len(commonResultError)):
51                         if commonResultError[i] < minError and commonResultError[i] <
                               self.__maxRelError and rel[i]:
52                             minError = commonResultError[i]
53                             dep = i
54                     if dep != 3:
55                         self.__fileWriter.writeResult(p1, p2, self.__accesDependList[
                               dep])
56                         countdep+=1
```

```
57                    progressBar.updateProgress(countdep)
```

Код *TestGenerator*

```
1  import random
2  from cmath import *
3
4  import numpy as np
5
6
7  class RandomDependGenerator:
8      def __init__(self, __countGenParameters, __maxvalue, __randomError):
9          self.__maxvalue=__maxvalue
10         self.__randomError=__randomError
11         self.__currentGenParameters = [[1.0, 2.0, 3.0]]
12         self.__countGenParameters = __countGenParameters
13
14
15     def randomGeneratedDepend(self):
16         for i in range(self.__countGenParameters-1):
17             fun = random.randint(0, 2)
18             if fun == 0:
19                 p = self.__expGen()
20             elif fun == 1:
21                 p = self.__linGen()
22             else: p = self.__quadGen()
23             self.__currentGenParameters.append(p)
24         return self.__currentGenParameters
25
26     def __expGen(self):
27         a = random.randint(-self.__maxvalue*4, self.__maxvalue*4)/2.0
28         error = random.randint(-self.__randomError, self.__randomError)/100.0
29         numberparam1 = random.randint(0, len(self.__currentGenParameters)-1)
30         param1 = np.array(self.__currentGenParameters[numberparam1], dtype=np.
               float128)
31         param2 = list(map(lambda x: float(np.round(((a*np.exp(x)).real + (a*np.exp(x)
               *error).real), 3)), param1))
32         return param2
33
34     def __linGen(self):
35         a = random.randint(-self.__maxvalue*4, self.__maxvalue * 4) / 2.0
36         b = random.randint(-self.__maxvalue*4, self.__maxvalue * 4) / 2.0
37         error = random.randint(-self.__randomError, self.__randomError) / 100.0
38         numberparam1 = random.randint(0, len(self.__currentGenParameters) - 1)
39         param1 = np.array(self.__currentGenParameters[numberparam1], dtype=np.
               float128)
40         param2 = list(map(lambda x: float(np.round(((a * x + b) + (a * x +b) * error)
               , 3)), param1))
41         return param2
42
43
44     def __quadGen(self):
45         a = random.randint(-self.__maxvalue*4, self.__maxvalue * 4) / 2.0
46         error = random.randint(-self.__randomError, self.__randomError) / 100.0
47         numberparam1 = random.randint(0, len(self.__currentGenParameters) - 1)
48         param1 = np.array(self.__currentGenParameters[numberparam1], dtype=np.
               float128)
49         param2 = list(map(lambda x: float(np.round((a*x*x + a*x*x*error), 3)), param1
               ))
50         return param2
```

Пакет GoogleTableHandler

Код *AutManager.py*

```
import json

import gspread
import getpass
from oauth2client.service_account import ServiceAccountCredentials


def authorized():
    while True:
        autkey = input("Введите путьдоключааутентификации    : ")
        scope = ['https://spreadsheets.google.com/feeds', 'https://www.googleapis.com
            /auth/drive']
        try:
            creds = ServiceAccountCredentials.from_json_keyfile_name(autkey, scope)
            client = gspread.authorize(creds)
            print("[Google Sheets] Authentification Complete!")
            return client
        except FileNotFoundError:
            print("Файл "+ autkey + " ненайден !")
        except json.decoder.JSONDecodeError:
            print("Файл неявляетсяключом  !")
            continue
```

Код *TableManager.py*

```
import gspread
import re

from numpy.core import float128


class TableManager:

    def __init__(self, __client, __tablename):
        self.__countParameters = 0
        self.__countMetrics = 0
        self.__tablename = __tablename
        self.__client = __client
        self.__table = self.__openTable()

    def __getTableListfromEdit(self):
        tableList = self.__table.get_all_values()
        return tableList

    def __openTable(self):
        try:
            if re.match("http", self.__tablename) :
                table=self.__client.open_by_url(self.__tablename).sheet1
            else:
                table = self.__client.open(self.__tablename).sheet1
        except gspread.exceptions.SpreadsheetNotFound:
            print("[ERROR]: You can not open this GoogleSheets!")
            exit(5)
        return table

    def getParamList(self):
        tableList= self.__getTableListfromEdit()
        paramList = []
        countMetrics = 0
        for parameter in tableList:
            if re.match("Parameter", parameter[0]):
```

```
37                  if (countMetrics != 0) and (len(parameter[1:]) != countMetrics):
38                      print("Count parameter error!")
39                      # TODO: ErrorHandler
40                      exit(6)
41                  paramList.append(list(map(float128, parameter[1:])))
42                  countMetrics = len(paramList[0])
43          return paramList
44
45      def setParamList(self, list):
46          self.__table.clear()
47          for i in range(len(list)):
48              self.__table.delete_row(i+1)
49              self.__table.insert_row((["Parameter " + str(i+1)] + list[i]), i+1)
```

Пакет Output

Код *ConsoleProgressBar.py*

```
import  sys
class ConsoleProgressBar:
    def __init__(self, sizeLine, countIteration):
        self.__currentSizeBar = 0
        self.__currentProgress = 0
        self.__sizeLine = sizeLine
        self.__countIteration = countIteration
        self.__sizeIterable = self.__sizeLine / self.__countIteration

    def startProcess(self):
        print("[DependFinder] Solver Progress:")

    def updateProgress(self, countDepend):
        self.__currentProgress += 1
        self.__currentSizeBar = round(1.0*self.__currentProgress * self.
            __sizeIterable)

        self.__bar = u'' * self.__currentSizeBar + ' ' * (self.__sizeLine - self.
            __currentSizeBar)

        percents = round(100.0 * self.__currentProgress / float(self.__countIteration
            ), 1)
        sys.stdout.write('[%s] %s%s ...%s\r' % (self.__bar, percents, '%', "
            Найденозависимостей : "+str(countDepend)+" из "+ str(self.__currentProgress)
            + " проверенных. "))
        sys.stdout.flush()
```

Код *OutputFileWriter.py*

```
class OutputFileWriter:
    def __init__(self, filenameOut):
        self.__filenameOut = filenameOut
        self.__file = open(self.__filenameOut, "w")

    def writeResult(self, NumberParam1, NumberParam2, type):
        self.__file.write("Параметр " + str(NumberParam1) + " связанспараметром    " +
            str(NumberParam2) +": Зависимость "+ type + "\n")
```

Вывод

В данной работе была написана программа для поиска функциональных связей между параметрами. Был использован метод наименьших квадратов ввиду его простоты и практичности. Были выведены все формулы для аппроксимации, а также для тестирования приложения был написан генератор случайных зависимостей.