МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ижевский государственный технический университет имени М. Т. Калашникова»

Институт «Информатика и вычислительная техника»

Кафедра «Программное обеспечение»

Работа защищена с оценкой

«__________»

Дата ____________

Подпись_________/__________

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к практике «Научно-исследовательская работа»

Выполнил

студент гр. Б18-191-2 А.Р. Рязапов

Руководитель

к.т.н., доцент А.Г. Русских

Рецензия:

степень достижения поставленной цели работы_____________________________________

полнота разработки темы____

уровень самостоятельности работы обучающегося_________________________________

недостатки работы____

____

Ижевск 2021

1. ЦЕЛЬ РАБОТЫ

Освоение и приобретение навыков решения научно-исследовательских задач с использованием современных инструментов реализованных в языке Python и его библиотеках, закрепление и углубление знаний, полученных обучающимися при теоретическом обучении, подготовка к изучению последующих дисциплин и прохождению других видов практики.

2. ЗАДАЧИ ПРАКТИКИ

1) Знакомство с задачами машинного обучения и анализа данных.

2) Знакомство с инструментами Python для решения научно-исследовательских задач.

3) Использование языка Python для решения математических задач.

4) Использование библиотек Python для задач линейной алгебры.

5) Получение навыков решения задач оптимизация и матричных разложений на Python.

6) Знакомство с задачами по теории вероятностей и математической статистики на Python.

3. ВЫПОЛНЕНИЕ ПРАКТИКИ

1. Язык Python и математика

Задачи этапа:

- Подготовить программную среду для работы.
- Написать простейшие скрипты на языке Python.
- Запомнить основные конструкции языка Python.
- Дать определение функции, предела, производной.
- Соотнести понятия производной, касательной и экстремума.

Задания этапа по программированию:

2.1.1. Установка среды программирования для Python

Задание заключается в установке Python и библиотек.

Задание 1:

1) Перейдите на сайт continuum.io по ссылке https://www.continuum.io/downloads

2) Выберете подходящую операционную систему и перейдите в соответствующий раздел сайта.

3) Следуя инструкциям на сайте, установите Python 2.7.

Задание 2:

1) Запустите iPython notebook. Команда запуска может несколько отличаться от команды запуска в инструкции из видео. Например, команда может быть такой: ipython-2.7 notebook

2) Создайте новый файл типа .ipynb

3) Создайте новую ячейку. В ней импортируйте библиотеку numpy (import numpy) и выведите на экран версию библиотеки (numpy.__version__)

4) Создайте новую ячейку. В ней импортируйте библиотеку scipy (import scipy) и выведите на экран версию библиотеки (scipy.__version__)

5) Создайте новую ячейку. В ней импортируйте библиотеку pandas (import pandas) и выведите на экран версию библиотеки (pandas.__version__)

6) Создайте новую ячейку. В ней импортируйте библиотеку matplotlib (import matplotlib) и выведите на экран версию библиотеки (matplotlib.__version__)

7) Сделайте скриншот №1, на котором будет хорошо видно результаты вашей работы, и загрузите его в форму.

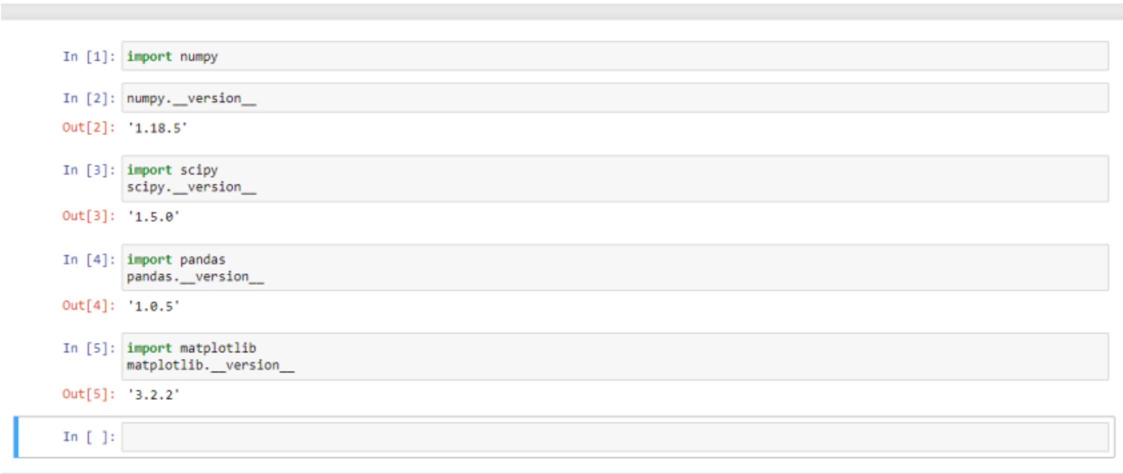


Рисунок 1. Результат выполнения задания 2

Задание 3:

1) Запустите iPython notebook. Команда запуска может несколько отличаться от команды запуска в инструкции, например, команда может быть такой ipython-2.7 notebook

2) Создайте новый файл типа .ipynb

3) В файле создайте новую ячейку и измените её тип на Markdown

4) В первой строке созданной ячейки наберите название специализации «Машинное обучение и анализ данных» и сделайте эту строку заголовком уровня 1

5) В следующей строке созданной ячейки наберите название нашего курса «Математика и Python» и сделайте строку заголовком уровня 2

6) В третьей строке ячейки наберите текст «Задание 1».

7) Запустите выполнение ячейки.

8) Сделайте скриншот №2, на котором будет хорошо видно результаты вашей работы, и загрузите его в форму.

В результате работы вы установите на компьютер Python и библиотеки, необходимые для дальнейшего прохождения курса.

Вы также научитесь запускать iPython notebook и выполнять в нем простые команды. Выполнение задания будет проверяться на основе двух скриншотов, которые вы прикрепите к заданию.

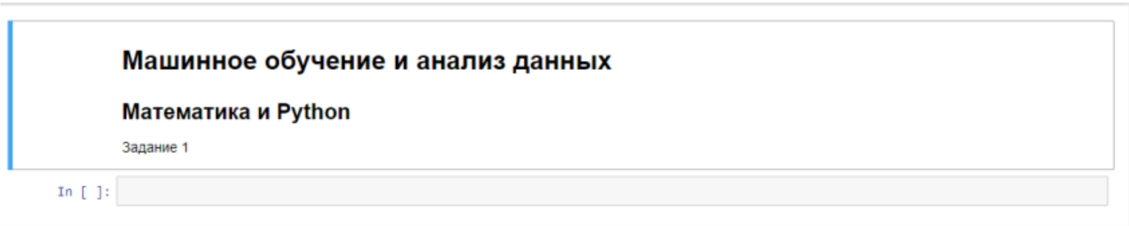


Рисунок 2. Результат выполнения задания 3

2.2. Библиотеки Python и линейная алгебра

Задачи этапа:

- Использовать средства модуля Pandas для загрузки данных и ознакомления с ними.
- Запомнить основные функции работы с датафреймами в Pandas.
- Использовать функции модуля numpy для выполнения операций с матрицами и векторами.
- Использовать функции модуля SciPy для решения математических задач (оптимизация функций, задачи линейной алгебры).
- Построить графики различных типов.
- Перечислить базовые понятия линейной алгебры.
- Соотнести понятия линейной независимости и размерности линейного пространства.
- Соотнести понятия нормы, длины, скалярного произведения, расстояния.
- Дать определения произведения матриц; ранга, определителя матрицы; собственных чисел и векторов матрицы; системы линейных уравнений.
- Соотнести понятия ранг, определитель матрицы и разрешимость системы линейных уравнений.

- Использовать средства Python для выполнения простых действий с текстами.

- Повторить элементы школьной математики (операции с числами, решение уравнений, многочлены, базовые понятия геометрии).

Задания этапа по программированию:

2.2.1. Линейная алгебра: сходство текстов и аппроксимация функций

Данное задание основано на материалах секции, посвященной введению в линейную алгебру. Вам понадобится компьютер с установленным интерпретатором Python и подключенными библиотеками NumPy и SciPy.

Данное задание состоит из двух частей. В каждой ответом будет набор чисел, который вам нужно будет ввести в соответствующее поле через пробел.

Задача 1: Сравнение предложений

Дан набор предложений, скопированных с Википедии. Каждое из них имеет «кошачью тему» в одном из трех смыслов:

- кошки (животные);
- UNIX-утилита cat для вывода содержимого файлов;
- версии операционной системы OS X, названные в честь семейства кошачьих.

Ваша задача - найти два предложения, которые ближе всего по смыслу к расположенному в самой первой строке. В качестве меры близости по смыслу мы будем использовать косинусное расстояние.

Выполните следующие шаги:

1) Скачайте файл с предложениями (sentences.txt).

2) Каждая строка в файле соответствует одному предложению. Считайте их, приведите каждую к нижнему регистру с помощью строковой функции lower().

3) Произведите токенизацию, то есть разбиение текстов на слова. Для этого можно воспользоваться регулярным выражением, которое считает разделителем любой символ, не являющийся буквой: re.split('[^a-z]', t). Не забудьте удалить пустые слова после разделения.

4) Составьте список всех слов, встречающихся в предложениях. Сопоставьте каждому слову индекс от нуля до (d - 1), где d — число различных слов в предложениях. Для этого удобно воспользоваться структурой dict.

5) Создайте матрицу размера n * d, где n — число предложений. Заполните ее: элемент с индексом (i, j) в этой матрице должен быть равен количеству вхождений j-го слова в i-е предложение. У вас должна получиться матрица размера 22 * 254.

6) Найдите косинусное расстояние от предложения в самой первой строке (In comparison to dogs, cats have not undergone...) до всех остальных с помощью функции scipy.spatial.distance.cosine. Какие номера у двух предложений, ближайших к нему по этому расстоянию (строки нумеруются с нуля)? Эти два числа и будут ответами на задание. Само предложение (In comparison to dogs, cats have not undergone...) имеет индекс 0.

7) Запишите полученные числа в файл, разделив пробелом. Обратите внимание, что файл должен состоять из одной строки, в конце которой не должно быть переноса. Пример файла с решением вы можете найти в конце задания (submission-1.txt).

8) Совпадают ли ближайшие два предложения по тематике с первым? Совпадают ли тематики у следующих по близости предложений?

Код программы к заданию 1:

```
import re
import numpy as np
file_obj = open('C:/Users/Amir/sentences.txt', 'r')
data_list = file_obj.readlines()
words = {}
data_list_ch =[]
for line in data_list:
    line = line.lower()
    line = re.split('[^a-z]', line)
    line = [x for x in line if x]
    for i in line:
        if i in words:
            words[i] +=1
        else:
            words[i] = 1
    data_list_ch.append(line)
counter = []
counter1 = []
for r in range(len(data_list_ch)):
    for i in words:
        counter.append(data_list_ch[r].count(i))
    counter1.append([])
    for c in range(len(counter)):
        counter1[r].append(counter[c])
    del counter[:]
from scipy.spatial import distance
length = len(counter1)
cos = []
for k in counter1:
    if k != counter1[0]:
        cos.append(distance.cosine(counter1[0], k))
fin = cos[:]
fin.sort()
```

```
file_obj_second = open('C:/Users/Amir/output.txt', 'a')
for i in range(len(cos)):
    if cos[i] == fin[0]:
        file_obj_second.write(str(i+1))
        file_obj_second.write(" ")
for i in range(len(cos)):
    if cos[i] == fin[1]:
        file_obj_second.write(str(i+1))
file_obj_second.close()
```

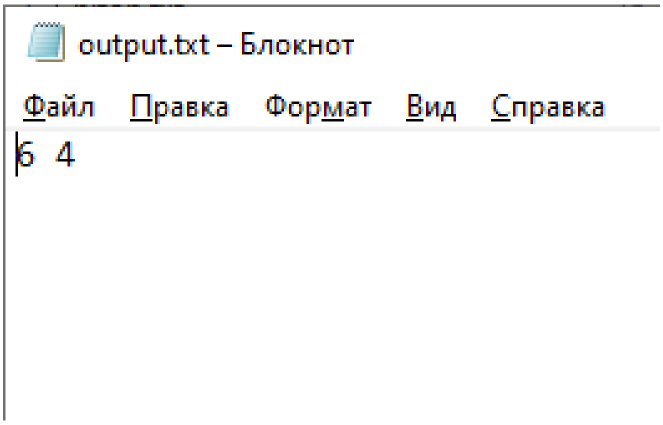


Рисунок 3. Файл output.txt

Задача 2: Аппроксимация функции

Рассмотрим сложную математическую функцию на отрезке [1, 15]:

$f(x) = \sin(x / 5) * \exp(x / 10) + 5 * \exp(-x / 2)$

1) Сформируйте систему линейных уравнений (то есть задайте матрицу коэффициентов A и свободный вектор b) для многочлена первой степени, который должен совпадать с функцией f в точках 1 и 15. Решите данную систему с помощью функции scipy.linalg.solve. Нарисуйте функцию f и полученный многочлен. Хорошо ли он приближает исходную функцию?

2) Повторите те же шаги для многочлена второй степени, который совпадает с функцией f в точках 1, 8 и 15. Улучшилось ли качество аппроксимации?

3) Повторите те же шаги для многочлена третьей степени, который совпадает с функцией f в точках 1, 4, 10 и 15. Хорошо ли он аппроксимирует функцию?

Коэффициенты данного многочлена (четыре числа в следующем порядке: w0, w1, w2, w3) являются ответом на задачу. Округлять коэффициенты не обязательно, но при желании можете произвести округление до второго знака (т.е. до числа вида 0.42).

4) Запишите полученные числа в файл, разделив пробелами. Обратите внимание, что файл должен состоять из одной строки, в конце которой не должно быть переноса.

Пример файла с решением вы можете найти в конце задания (submission-2.txt).

Код программы к заданию 2:

```
import scipy.linalg
import numpy as np
import matplotlib.pyplot as plt
import math

# Функция
function_x = np.linspace(1, 15)
function_y = []
for i in function_x:
    function_y.append(math.sin(i / 5) * math.exp(i / 10) + 5 * math.exp(-i / 2))

# Линейное приближение
first_degree = [1, 15]
A = []
for i in first_degree:
    vector = []
    for j in range (len(first_degree)):
        vector.append(i ** j)
    A.append(vector)
b = []
for i in first_degree:
    b.append(math.sin(i / 5) * math.exp(i / 10) + 5 * math.exp(-i / 2))
array_of_solutions = scipy.linalg.solve(A, b)

polynomial_y = []
for i in function_x:
    y = 0
    for j in range(len(array_of_solutions)):
        y += ((i ** j) * array_of_solutions[j])
    polynomial_y.append(y)

plt.figure(1)
plt.title("Функция f(x) и многочлен первой степени")
plt.plot(function_x, polynomial_y, function_x, function_y)

# Квадратичное приближение
second_degree = [1, 8, 15]
A.clear()
for i in second_degree:
    vector = []
    for j in range (len(second_degree)):
        vector.append(i ** j)
    A.append(vector)
b.clear()
for i in second_degree:
    b.append(math.sin(i / 5) * math.exp(i / 10) + 5 * math.exp(-i / 2))
array_of_solutions = scipy.linalg.solve(A, b)

polynomial_y.clear()
for i in function_x:
    y = 0
```

```
    for j in range(len(array_of_solutions)):
        y += ((i ** j) * array_of_solutions[j])
    polynomial_y.append(y)

plt.figure(2)
plt.title("Функция f(x) и многочлен второй степени")
plt.plot(function_x, polynomial_y, function_x, function_y)

# Кубическое приближение
third_degree = [1, 4, 8, 15]
A.clear()
for i in third_degree:
    vector = []
    for j in range (len(third_degree)):
        vector.append(i ** j)
    A.append(vector)
b.clear()
for i in third_degree:
    b.append(math.sin(i / 5) * math.exp(i / 10) + 5 * math.exp(-i / 2))
array_of_solutions = scipy.linalg.solve(A, b)

polynomial_y.clear()
for i in function_x:
    y = 0
    for j in range(len(array_of_solutions)):
        y += ((i ** j) * array_of_solutions[j])
    polynomial_y.append(y)

plt.figure(3)
plt.title("Функция f(x) и многочлен третей степени")
plt.plot(function_x, polynomial_y, function_x, function_y)

file_obj = open("output2.txt", "a")
file_obj.write(str(round(array_of_solutions[0], 2)))
for i in range (len(array_of_solutions)):
    if i != 0:
        file_obj.write(" ")
        file_obj.write(str(round(array_of_solutions[i], 2)))
file_obj.close()
```

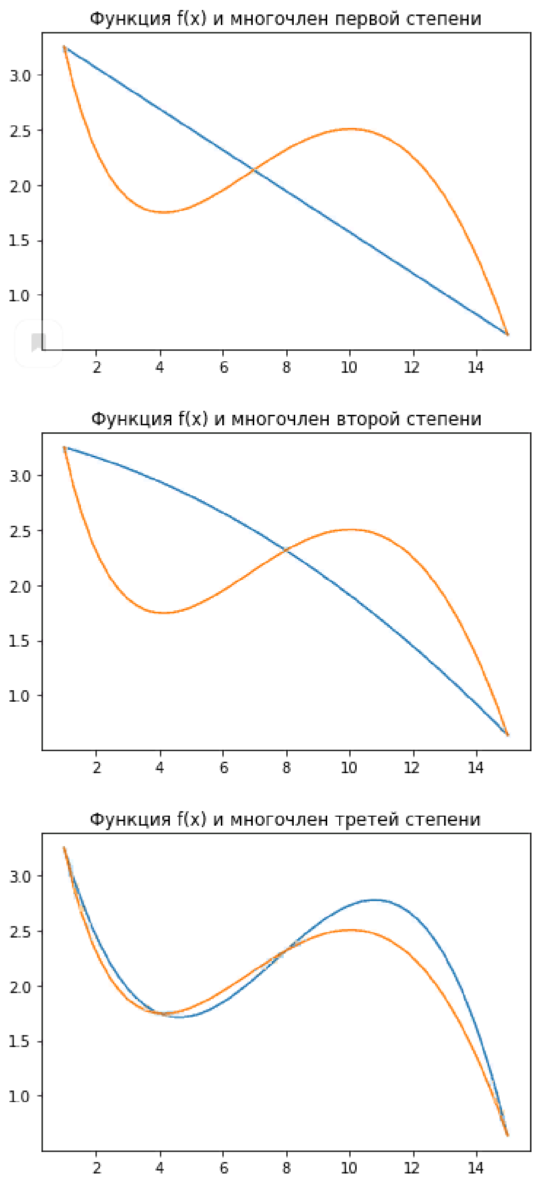
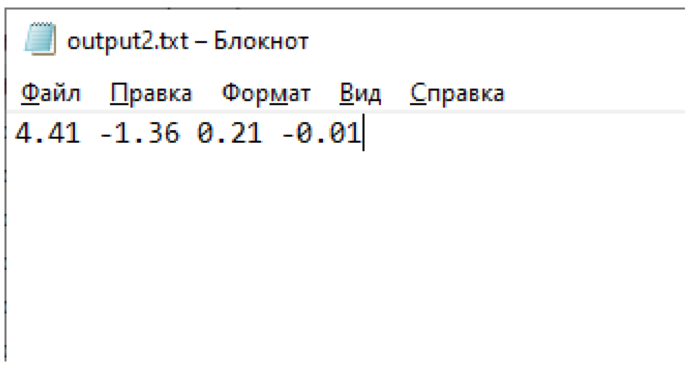


Рисунок 4. Результат выполнения задания 2



4.41 -1.36 0.21 -0.01

Рисунок 5. Файл output2.txt

2.3. Оптимизация и матричные разложения на Python

Задачи этапа:

- Соотнести понятия частной производной, градиента, производной по направлению.
- Перечислить свойства градиента функции.
- Объяснить принцип работы метода градиентного спуска.
- Резюмировать недостатки метода градиентного спуска.
- Разобрать примеры методов оптимизации негладкой функции.
- Объяснить принципы работы метода имитации отжига, дифференциальной эволюции, метода Нелдера-Мида.
- Выбрать метод оптимизации в конкретной задаче.
- Применить методы оптимизации, реализованные в SciPy.
- Объяснить связь между сингулярным разложением и приближением матрицей меньшего ранга.
- Разобрать примеры использования матричных разложений при решении задач анализа данных.
- Сформулировать задачу приближения матрицы матрицей меньшего ранга

Задание этапа по программированию:

2.3.1. Оптимизация в Python: глобальная оптимизация и оптимизация негладкой функции

Данное задание основано на материалах секции, посвященной оптимизационным задачам и методам их решения. Вам понадобится компьютер с установленным интерпретатором Python и подключенными библиотеками NumPy, SciPy и Matplotlib.

Инструкция по выполнению. Данное задание состоит из трех частей. В каждой ответом будет набор чисел, который вам нужно будет набрать через пробел в текстовом файле и загрузить. Десятичные дроби записывайте через точку.

Задача 1. Минимизация гладкой функции

1) Рассмотрим все ту же функцию из задания по линейной алгебре:

f(x) = sin(x / 5) * exp(x / 10) + 5 * exp(-x / 2), но теперь уже на промежутке [1, 30]

2) В первом задании будем искать минимум этой функции на заданном промежутке с помощью scipy.optimize. Разумеется, в дальнейшем вы будете использовать методы оптимизации для более сложных функций, а f(x) мы рассмотрим как удобный учебный пример.

3) Напишите на Python функцию, вычисляющую значение f(x) по известному x. Будьте внимательны: не забывайте про то, что по умолчанию в питоне целые числа делятся нацело, и о том, что функции sin и exp нужно импортировать из модуля math.

4) Изучите примеры использования scipy.optimize.minimize в документации Scipy (см. «Материалы»).

5) Попробуйте найти минимум, используя стандартные параметры в функции scipy.optimize.minimize (т.е. задав только функцию и начальное приближение). Попробуйте менять начальное приближение и изучить, меняется ли результат.

6) Укажите в scipy.optimize.minimize в качестве метода BFGS (один из самых точных в большинстве случаев градиентных методов оптимизации), запустите из начального приближения x=2. Градиент функции при этом указывать не нужно – он будет оценен численно. Полученное значение функции в точке минимума - ваш первый ответ по заданию 1, его надо записать с точностью до 2 знака после запятой.

7) Теперь измените начальное приближение на x=30. Значение функции в точке минимума - ваш второй ответ по заданию 1, его надо записать через пробел после первого, с точностью до 2 знака после запятой.

8) Стоит обдумать полученный результат. Почему ответ отличается в зависимости от начального приближения? Если нарисовать график функции (например, как это делалось в видео, где мы знакомились с Numpy, Scipy и Matplotlib), можно увидеть, в какие именно минимумы мы попали. В самом деле, градиентные методы обычно не решают задачу глобальной оптимизации, поэтому результаты работы ожидаемые и вполне корректные.

Код программы к заданию 1:

```
import scipy.optimize
import numpy as np
import matplotlib.pyplot as plt
import math

def f(x):
    return math.sin(x / 5) * math.exp(x / 10) + 5 * math.exp(-x / 2)

two = scipy.optimize.minimize(f, 2, method="BFGS")
thirty = scipy.optimize.minimize(f, 30, method="BFGS")
print("%.2f" % two.fun, end = " ")
print("%.2f" % thirty.fun)
function_x = np.linspace(1, 30)
function_y = []
for i in function_x:
```

```
    function_y.append(f(i))
plt.plot(function_x, function_y)
```

Вывод программы:

```
1.75 -11.90
```

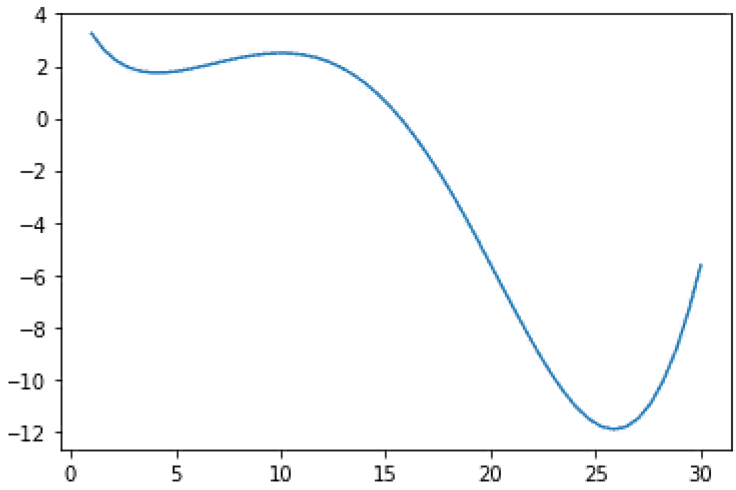


Рисунок 6. Результат выполнения программы

Задача 2. Глобальная оптимизация

1) Теперь попробуем применить к той же функции f(x) метод глобальной оптимизации - дифференциальную эволюцию.

2) Изучите документацию и примеры использования функции scipy.optimize.diierential_evolution.

3) Обратите внимание, что границы значений аргументов функции представляют собой список кортежей (list, в который помещены объекты типа tuple). Даже если у вас функция одного аргумента, возьмите границы его значений в квадратные скобки, чтобы передавать в этом параметре список из одного кортежа, т.к. в реализации scipy.optimize.diierential_evolution длина этого списка используется чтобы определить количество аргументов функции.

4) Запустите поиск минимума функции f(x) с помощью дифференциальной эволюции на промежутке [1, 30]. Полученное значение функции в точке минимума - ответ в задаче 2. Запишите его с точностью до второго знака после запятой. В этой задаче ответ - только одно число.

5) Заметьте, дифференциальная эволюция справилась с задачей поиска глобального минимума на отрезке, т.к. по своему устройству она предполагает борьбу с попаданием в локальные минимумы.

6) Сравните количество итераций, потребовавшихся BFGS для нахождения минимума при хорошем начальном приближении, с количеством итераций, потребовавшихся дифференциальной эволюции. При повторных запусках

дифференциальной эволюции количество итераций будет меняться, но в этом примере, скорее всего, оно всегда будет сравнимым с количеством итераций BFGS. Однако в дифференциальной эволюции за одну итерацию требуется выполнить гораздо больше действий, чем в BFGS. Например, можно обратить внимание на количество вычислений значения функции (nfev) и увидеть, что у BFGS оно значительно меньше. Кроме того, время работы дифференциальной эволюции очень быстро растет с увеличением числа аргументов функции.

Код программы к заданию 2:

```
import scipy.optimize
import math

def f(x):
    return math.sin(x / 5) * math.exp(x / 10) + 5 * math.exp(-x / 2)

dif_evol = scipy.optimize.differential_evolution(f, [(1, 30)])
minim = scipy.optimize.minimize(f, 30, method="BFGS")
print("%.2f" % dif_evol.fun)
print("Дифференциальная эволюция:", dif_evol.nit)
print("BFGS:", minim.nit)
print("Количество вычислений значения функции в дифференциальной эволюции:",
dif_evol.nfev)
print("Количество вычислений значения функции в BFGS:", minim.nfev)
```

Вывод программы:

```
-11.90
Дифференциальная эволюция: 5
BFGS: 6
Количество вычислений значения функции в дифференциальной эволюции: 96
Количество вычислений значения функции в BFGS: 14
```

Задача 3. Минимизация негладкой функции

1) Теперь рассмотрим функцию h(x) = int(f(x)) на том же отрезке [1, 30], т.е. теперь каждое значение f(x) приводится к типу int и функция принимает только целые значения.

2) Такая функция будет негладкой и даже разрывной, а ее график будет иметь ступенчатый вид. Убедитесь в этом, построив график h(x) с помощью matplotlib.

3) Попробуйте найти минимум функции h(x) с помощью BFGS, взяв в качестве начального приближения x=30. Получившееся значение функции – ваш первый ответ в этой задаче.

4) Теперь попробуйте найти минимум h(x) на отрезке [1, 30] с помощью дифференциальной эволюции. Значение функции h(x) в точке минимума – это ваш второй ответ в этом задании. Запишите его через пробел после предыдущего.

5) Обратите внимание на то, что полученные ответы различаются. Это ожидаемый результат, ведь BFGS использует градиент (в одномерном случае – производную) и явно не пригоден для минимизации рассмотренной нами разрывной функции. Попробуйте понять, почему минимум, найденный BFGS, именно такой (возможно в этом вам поможет выбор разных начальных приближений).

6) Выполнив это задание, вы увидели на практике, чем поиск минимума функции отличается от глобальной оптимизации, и когда может быть полезно применить вместо градиентного метода оптимизации метод, не использующий градиент. Кроме того, вы попрактиковались в использовании библиотеки SciPy для решения оптимизационных задач, и теперь знаете, насколько это просто и удобно.

Код программы к заданию 3:

```
import scipy.optimize
import numpy as np
import matplotlib.pyplot as plt
import math

def h(x):
    return int(math.sin(x / 5) * math.exp(x / 10) + 5 * math.exp(-x / 2))

x = np.linspace(1, 30)
y = []
for i in x:
    y.append(h(i))
plt.plot(x, y)
minim = scipy.optimize.minimize(h, 30, method="BFGS")
print(minim.fun, end = " ")
dif_evol = scipy.optimize.differential_evolution(h, [(1, 30)])
print(dif_evol.fun)
```

Вывод программы:

```
-5 -11.0
```

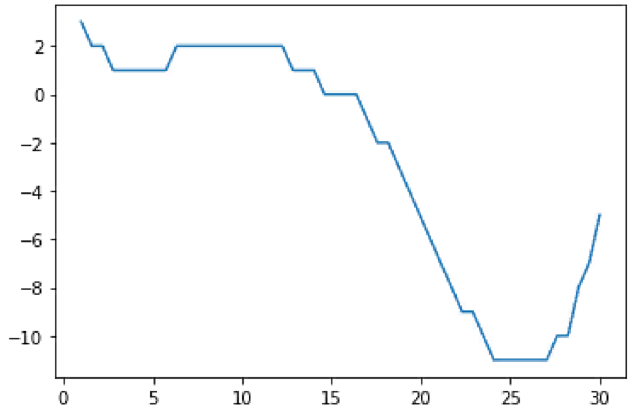


Рисунок 7. Результат выполнения программы

2.4. Теория вероятностей и математическая статистика на Python

Задачи этапа:

- Перечислить свойства вероятности.
- Сформулировать определение условной вероятности, формулу полной вероятности и
формулу Байеса.
- Вычислить вероятность события в задаче.
- Классифицировать случайные величины.
- Объяснить разницу между непрерывной и дискретной случайной величиной.
- Дать примеры дискретных и непрерывных случайных величин.
- Различить функцию распределения и плотность распределения.
- Перечислить характеристики распределений и понимать, какие свойства распределения
они описывают.
- Перечислить основные статистики распределений и понимать, какие свойства
распределения они описывают.
- Сформулировать центральную предельную теорему (ЦПТ).
- Продемонстрировать применение ЦПТ для построения доверительного интервала.
- Различить характеристики распределения и статистики распределения, оцененные по выборке.
- Построить гистограмму распределения.

Задания этапа по программированию:

2.4.1. Центральная предельная теорема своими руками

В этом задании вам предстоит проверить работу центральной предельной теоремы, а также поработать с генерацией случайных чисел и построением графиков в Python.

Выберите ваше любимое непрерывное распределение (чем меньше оно будет похоже на нормальное, тем интереснее; попробуйте выбрать какое-нибудь распределение из тех, что мы не обсуждали в курсе). Сгенерируйте из него выборку объёма 1000, постройте гистограмму выборки и нарисуйте поверх неё теоретическую плотность распределения вашей случайной величины (чтобы величины были в одном масштабе, не забудьте выставить у гистограммы значение параметра normed=True).

Ваша задача - оценить распределение выборочного среднего вашей случайной величины при разных объёмах выборок. Для этого при трёх и более значениях n (например, 5, 10, 50) сгенерируйте 1000 выборок объёма n и постройте гистограммы распределений их выборочных средних. Используя информацию о среднем и дисперсии исходного распределения (её можно без труда найти в википедии), посчитайте значения параметров нормальных распределений, которыми, согласно центральной предельной теореме, приближается распределение выборочных средних. Обратите внимание: для подсчёта значений этих параметров нужно использовать именно теоретические среднее и дисперсию вашей случайной величины, а не их выборочные оценки. Поверх каждой гистограммы нарисуйте плотность соответствующего нормального распределения (будьте внимательны с параметрами функции, она принимает на вход не дисперсию, а стандартное отклонение). Опишите разницу между полученными распределениями при различных значениях n. Как меняется точность аппроксимации распределения выборочных средних нормальным с ростом n?

Код программы:

```
from scipy.stats import laplace
import scipy.stats as sts
import matplotlib.pyplot as plt
import numpy as np
laplace_rv=sts.laplace(0,1)
arrayValueRv=laplace_rv.rvs(size=1000)
plt.hist(arrayValueRv, density=True, label='гистограмма выборки')
x=np.linspace(-10,10,500)
denPdf=laplace_rv.pdf(x)
plt.plot(x, denPdf,lw=2,label='график плотности распределения')
plt.xlabel('x')
plt.ylabel('F(x)')
plt.legend(loc='upper right')
plt.show()
```

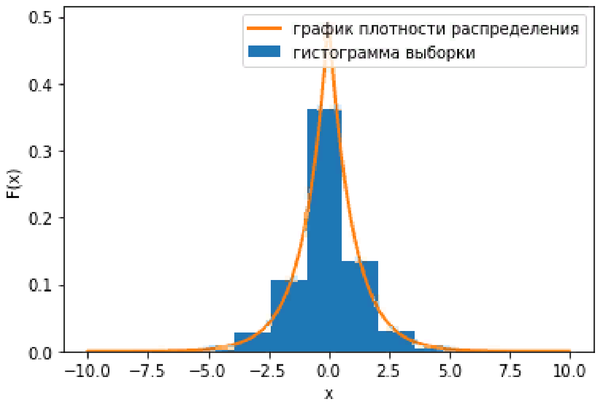


Рисунок 8. Результат выполнения программы

```
import math
valueMean=laplace_rv.mean()
disp=laplace_rv.var()
for n in [5,10,50]:
    val=np.linspace(-10,10,500)
    normValue=sts.norm(valueMean, disp/n)
    norm_pdf=normValue.pdf(val)
    res=[]
    for j in range(1000):
        res.append(laplace_rv.rvs(n).mean())
    plt.hist(res, density=True, label='гистограмма распределений выборочных средних')
    plt.plot(val, norm_pdf, label='плотность нормального распределения')
    plt.title(f'n = {n}')
    plt.xlabel('выборочное среднее')
    plt.ylabel('F(x)')
    plt.legend(loc='upper right')
    plt.show()
```

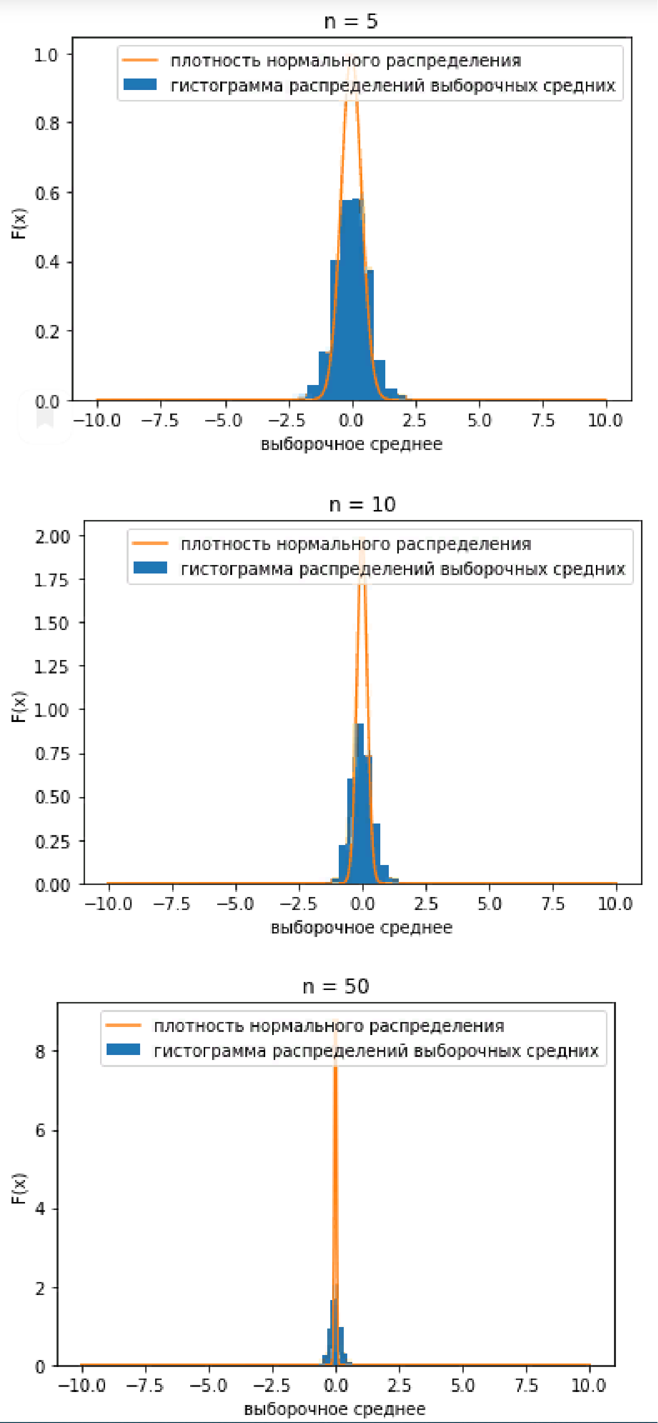


Рисунок 9. Результат выполнения программы

Вывод: с ростом числа n гистограмма становится похожей на нормальное распределение.

ЗАКЛЮЧЕНИЕ

В ходе практики освоены и приобретены навыки решения научно-исследовательских задач с использованием современных инструментов реализованных в языке Python и его библиотеках, закреплены и углублены знания, полученные при теоретическом обучении.