А. А. Дробышев,
П. А. Головинский,
Е. А. Михин

ОСНОВЫ МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ И ОПТИМИЗАЦИИ НА ЯЗЫКЕ PYTHON

Учебно-методическое пособие

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

А. А. Дробышев, П. А. Головинский, Е. А. Михин

ОСНОВЫ МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ И ОПТИМИЗАЦИИ НА ЯЗЫКЕ PYTHON

Учебно-методическое пособие

Москва
Знание-М
2023

УДК 004.94(075.3)
ББК 32.973.2
Д75

Рекомендовано научно-методическим советом
Воронежского государственного технического университета
в качестве учебного пособия для студентов,
обучающихся по направлениям 27.03.05 и 27.04.05 «Инноватика»,
38.04.01 «Экономика»

Рецензент:

Юдаков Д. С. – кандидат технических наук, доцент кафедры средств связи (и авиационных комплексов связи),
Военный учебно-научный центр Военно-воздушных сил
«Военно-воздушная академия имени профессора Н.Е. Жуковского и Ю.А.Гагарина» (г. Воронеж)
Министерства обороны Российской Федерации

Дробышев, Алексей Александрович.
Д75 Основы математического моделирования и оптимизации на языке Python : учебно-методическое пособие / А. А. Дробышев, П. А. Головинский, Е. А. Михин. – Москва : Знание-М, 2023. – 107 с.

ISBN 978-5-00187-654-0

В пособии представлены основные методы применения языка Python для решения задач моделирования, оптимизации и обработки данных. Каждая рассмотренная тема включает краткие теоретические сведения, подробный разбор практической работы, индивидуальные варианты заданий и контрольные вопросы по теме.

Пособие предназначено для студентов всех форм обучения по направлениям «Инноватика» и «Экономика», а также может использоваться при изучении языка программирования Python и в качестве компьютерного практикума для студентов различных специальностей, применяющих методы математического и компьютерного моделирования.

УДК 004.94(075.3)
ББК 32.973.2

ISBN 978-5-00187-654-0

Содержание

Введение

В настоящее время компьютеры и различные вычислительные системы стали неотъемлемой частью как бытовой жизни людей, так и их профессиональной деятельности. Развитие науки, экономики и промышленности, постоянный рост объема обрабатываемой информации и сложности решаемых задач не только увеличивает роль компьютерного моделирования, но и делает его незаменимым в современном мире.

Компьютер перестал быть инструментом только инженеров, ученых и программистов. Все больше задач, решение которых требует применения компьютерного моделирования, возникает перед экономистами, менеджерами, маркетологами, аналитиками, руководителями и другими специалистами различных профилей. Поэтому приобретение навыков программирования и компьютерного моделирования является актуальной задачей при обучении студентов различных специальностей.

Данное учебно-методическое пособие представляет собой компьютерный практикум по математическому моделированию на языке программирования Python.

В пособии для получения базовых навыков разработки компьютерных моделей и начального ознакомления с возможностями применения языка Python для решения задач математического моделирования и оптимизации представлены 8 практических работ. К каждой работе приведены краткие теоретические сведения по теме работы, включающие математические модели и основные элементы языка Python, а также пример решения типовой задачи. Все примеры выполнены на Python версии 3 в среде Jupyter Notebook.

Выполнение практической работы заключается в изучении студентами её теоретической части, в том числе необходимой литературы, и последовательном решении поставленных в работе задач.

Результат выполнения работы представляется посредством демонстрации работающей программы на языке Python и в виде оформленного отчета. Отчет содержит математическую модель задачи и ее решения, компьютерную модель, включая блок-схемы

алгоритмов и их программный код, а также все необходимые пояснения, результаты и выводы.

Построение компьютерной модели изучаемого явления или процесса состоит из нескольких последовательных взаимосвязанных этапов, среди которых обычно выделяют следующие:

а) постановка задачи;

б) построение математической модели задачи;

в) разработка или выбор метода решения;

г) разработка алгоритма численного решения задачи (алгоритмизация);

д) оптимизация алгоритма;

е) написание кода алгоритма на языке программирования;

ж) отладка кода;

з) проверка результатов и корректировка программы;

и) получение результатов для заданного диапазона параметров и их анализ.

Алгоритмом называют план выполнения какой-либо задачи, приводящий к поставленной цели. Любая компьютерная программа реализует некоторый алгоритм. Поэтому и работа программы аналогична работе алгоритма: последовательное выполнение предписаний по очереди с первой строки и до конца программы.

Этап разработки алгоритма заключается в разделении всего вычислительного процесса на отдельные составные части, установлении порядка их следования и описании содержания каждой части.

Процесс разработки алгоритма проходит несколько этапов детализации. На первом этапе составляется общая схема алгоритма, в которой отражаются наиболее важные его части и связи между ними. На следующем этапе происходит детализация выделенных на первом этапе частей, затем детализация этих частей и т.д. Каждая отдельная часть алгоритма должна быть самостоятельной и логически завершенной, что позволяет проводить дальнейшую её детализацию независимо от остальных частей.

Общая схема алгоритма позволяет разработать несколько его вариантов, провести их сравнение, анализ и выбрать оптимальный.

Существуют разные способы описания алгоритма. Наиболее распространенными являются словесный и графический способы.

Алгоритм должен обладать следующими свойствами.

1. Дискретность. Алгоритм имеет пошаговый характер. При этом должно выполняться условие: выполнение следующего предписания (правила или команды) возможно только после выполнения предыдущего.

2. Понятность для исполнителя. Это означает, что в алгоритме должны быть команды, которые известны исполнителю.

3. Однозначность. Алгоритм должен содержать команды, приводящие к однозначным действиям.

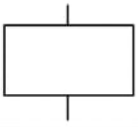
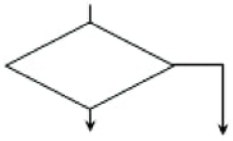
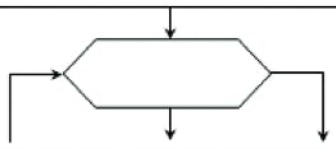
4. Массовость. Алгоритм должен быть пригоден для решения широкого круга задач данного типа, отличающихся только исходными данными.

5. Конечность. Алгоритм должен заканчиваться после выполнения конечного числа операций.

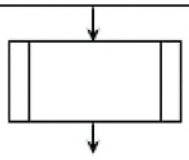
6. Алгоритм должен приводить к правильному результату.

При графической форме записи алгоритма каждое его предписание изображается в виде геометрической фигуры – блока. Переходы между блоками изображаются линиями, а направление перехода стрелками. Получившаяся в результате структура из блоков и связей между ними называется блок-схемой. Основные элементы блок-схем представлены в таблице В.1.

Таблица В.1 – Основные элементы блок-схемы

Обозначение	Название	Функция
	Вычислительный блок	Выполнение операций
	Логический блок	Выбор направления выполнения алгоритма в зависимости от условий
	Блок модификации	Объявление цикла

Продолжение таблицы В.1

	Начало-конец	Начало, прерывание и конец выполнения программы
	Ввод-вывод	Ввод и вывод данных
	Подпрограмма	Использование ранее созданных алгоритмов и программ
	Соединитель	Указание связи между прерванными линиями
	Межстраничный соединитель	Указание связи между частями алгоритма на разных листах

В качестве примера рассмотрим алгоритм решения квадратного уравнения. Решение этой задачи выполним следуя этапам разработки компьютерной модели.

Этап 1. Постановка задачи.

Формулируем условие и цель решаемой задачи: дано квадратное уравнение, найти его корни.

Этап 2. Построение математической модели задачи.

Сформулированную на первом этапе задачу записываем в математическом виде: дано квадратное уравнение

$$ax^2 + bx + c = 0 .$$

Найти все x, удовлетворяющие этому уравнению.

Этап 3. Разработка или выбор метода решения.

Если $a \neq 0$, то корни уравнения будем вычислять с помощью дискриминанта

$$D = b^2 - 4ac . \qquad \text{(В.1)}$$

Если $D \geq 0$, то

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}, \qquad \text{(В.2)}$$

если $D < 0$, то

$$x_{1,2} = \frac{-b \pm i\sqrt{|D|}}{2a}, \qquad \text{(В.3)}$$

если $a = 0$ и $b \neq 0$, то корень уравнения

$$x = -\frac{c}{b}. \qquad \text{(В.4)}$$

Этап 4. Разработка алгоритма решения задачи.

Начинаем со словесного алгоритма решения задачи:

а) Ввод коэффициентов a, b, c с клавиатуры.

Проверка условия $a \neq 0$. Если оно верно, то далее следует пункт «в», если не верно, то проверяется условие $b \neq 0$. Если последнее условие верно, то корень уравнения рассчитывается по формуле (В.4).

Далее выводится сообщение «корень уравнения равен x», после чего завершается программа.

Если условие $b \neq 0$ не верно, то происходит вывод сообщения «ошибка ввода данных» и завершение программы.

б) Расчет дискриминанта.

в) Проверка знака дискриминанта. Если он больше нуля, то корни рассчитываются по формуле (В.2), если меньше нуля, то корни рассчитываются по формуле (В.3).

г) Вывод сообщения «корни уравнения равны x_1 и x_2» и завершение программы.

Разработанный словесный алгоритм представим в виде блок-схемы (рис. В.1).

Этап 5. Оптимизация алгоритма.

Теперь проверяем наш алгоритм на возможность его упрощения. В данном случае таких существенных упрощений не наблюдается.

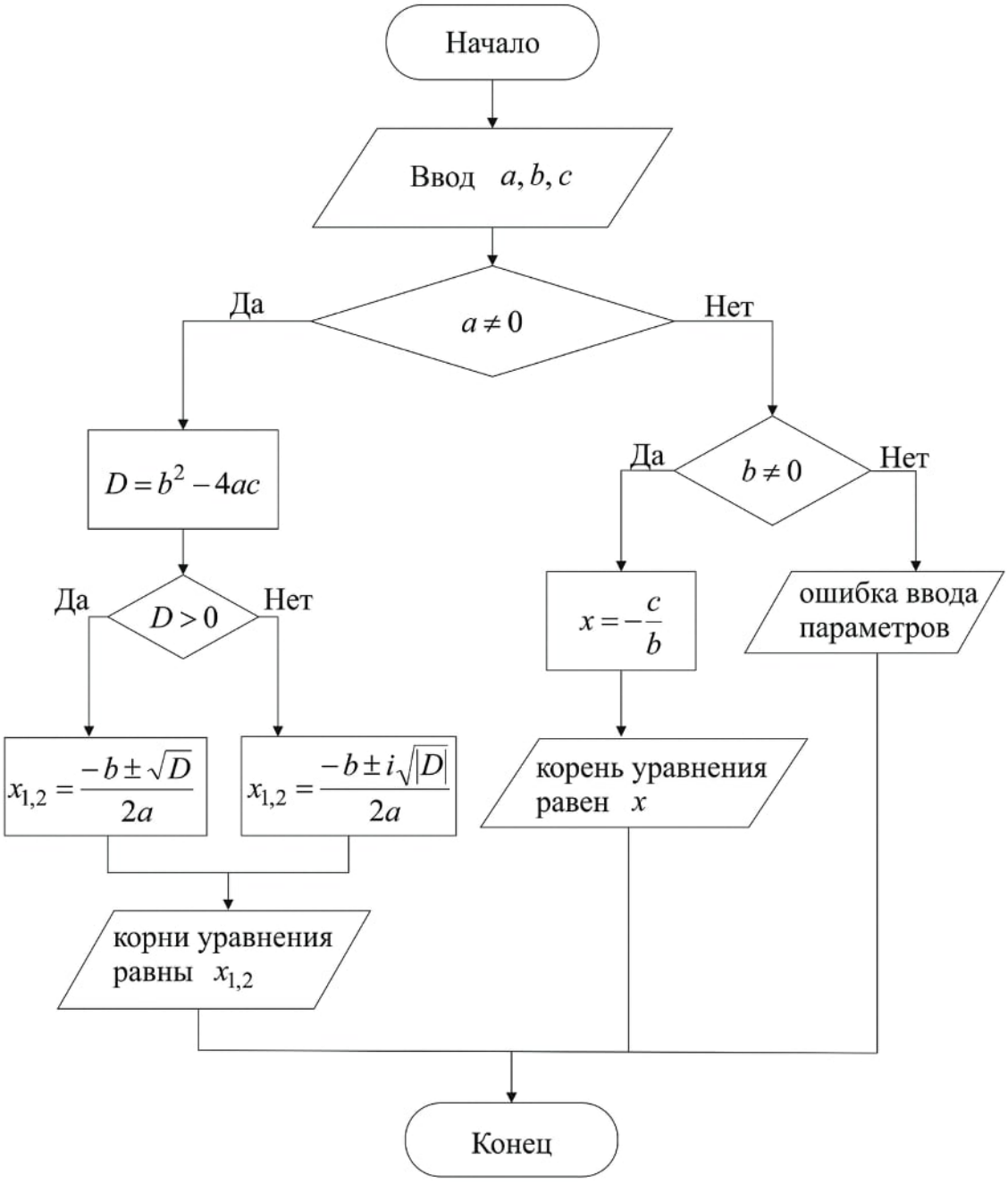


Рисунок В.1 – Блок-схема алгоритма решения квадратного уравнения

Этап 6. Написание кода алгоритма на языке программирования.

Этап 7. Отладка кода.

Это завершающий этап. Производится поиск ошибок и проверка работоспособности программы при разных значениях параметров.

На этом решение задачи заканчивается.

Тема №1. Моделирование и графическое представление функций в среде Python

1.1 Краткие теоретические сведения

Любая функция может быть записана с помощью арифметических действий и различных других функций. Основные арифметические операторы, применяющиеся в Python, приведены в таблице 1.1.

Таблица 1.1 – Основные арифметические операторы в Python

Оператор	Описание
a + b	Сложение a и b
a – b	Вычитание b из a
a*b	Умножение a на b
a**b	Возведение a в степень b
a/b	Деление a на b
a//b	Деление a на b без остатка (остаток отбрасывается)
a%b	Целый остаток от деления a на b

Для того чтобы присвоить результат вычисления какой-либо переменной, применяется оператор присвоения «=».

В Python представлены встроенные функции и дополнительные, собранные в различные библиотеки. Чтобы пользоваться этими дополнительными функциями, нужно перед их вызовом загрузить библиотеку, в которой находятся данные функции, или загрузить сами функции из библиотек. Библиотека (функция, модуль и т.д.) загружается с помощью команды `import` обычно под определенным сокращенным именем.

Например, загрузим библиотеку NumPy [1] под именем `np`, воспользуемся функцией `sin()` из этой библиотеки и выведем значение $\sin(\pi)$:

```
In[1]:
import numpy as np
print(np.sin(pi))

Out[1]:
0
```

Как видно, в Python перед именем вызываемой функции из некоторой библиотеки (модуля) записывается имя библиотеки, под которым мы ее загрузили.

Рассмотрим построение графика функции.

Одной из основных библиотек в Python для визуализации данных является библиотека Matplotlib [2]. Будем использовать для построения графика функцию `plot()` из модуля pyplot библиотеки Matplotlib.

Для построения графика функции необходимо создать набор значений аргумента. Этот набор можно создать, например, с помощью функции `arange(a,b,c)` из библиотеки NumPy. Функция `arange(a,b,c)` создает набор значений от `a` включительно, до `b` не включительно, с шагом `c`.

В качестве примера, построим график функции $\sin(x)$ на интервале от 0 до 20π, шаг аргумента 0.1 (пример 1.1).

Пример 1.1. Построение графика функции $\sin(x)$.

```
In[2]:
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 20*np.pi, 0.1)
plt.plot(x, np.sin(x), marker='*', color='r')
```

В этом примере мы использовали дополнительные возможности визуализации, доступные в функции `plot()`. В частности, в качестве точек графика использован маркер в виде звездочки «*», а его цвет выбран красным. Более подробно с многообразием стилей отображения графиков можно ознакомиться в литературе [2, 3].

Результат работы программного кода из примера 1.1 представлен на рисунке 1.1.

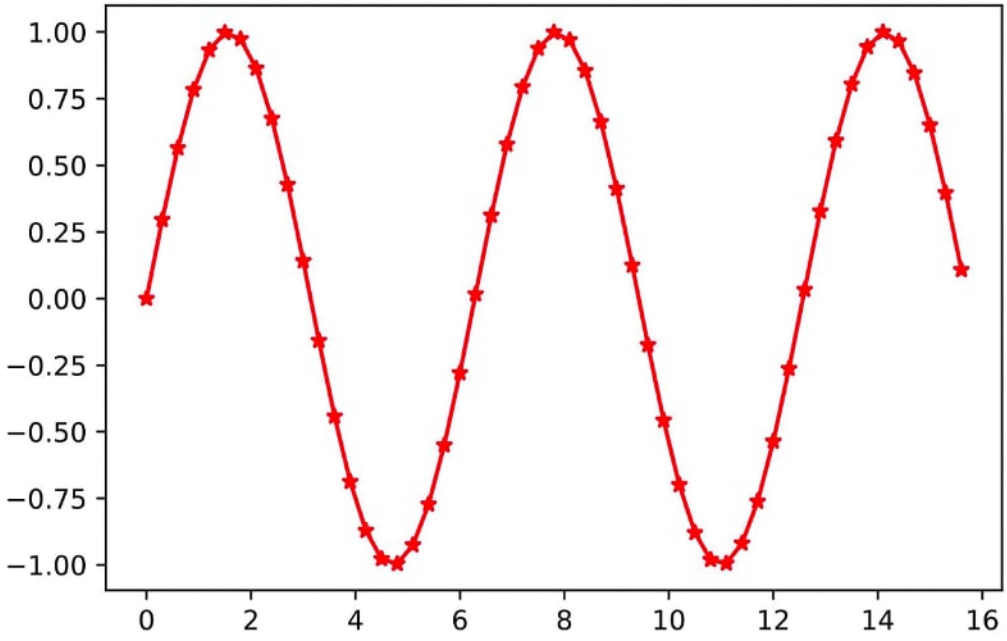


Рисунок 1.1 – Результат работы кода из примера 1.1

В примере 1.1 для размера рисунка и большинства других параметров применялись значения по умолчанию. Так, при выполнении команды `plt.plot()` создавался стандартный рисунок (`figure`), представляющий собой область для размещения графика (или графиков), стандартный график (`axes`), на котором располагаются оси координат (`axis`), кривая графика функции, подписи осей, различные обозначения и т.д. Все элементы рисунка можно настраивать вручную.

В Matplotlib возможно создание нескольких графиков на одном рисунке, например, с помощью методов `add_axes()` и `add_subplot()`. Для ручной расстановки графиков используется метод `add_axes()` для каждого из них, в котором указывается его расположение и размеры. Для автоматического разделения рисунка на нужное количество графиков используется метод `add_subplot(a,b,c)`. Этот метод представляет рисунок в виде матрицы («сетки»), в которой `a` – количество строк, `b` – количество столбцов, а `c` – номер графика (нумерация начинается с левой верхней ячейки).

При создании нескольких графиков на одном рисунке по умолчанию остается свободное пространство между ними.

Изменять это расстояние можно методом `tight_layout()`, имеющим параметры:

- `pad` – расстояние между краями рисунка (`figure`) и краями графиков;
- `h_pad` – расстояние по вертикали между графиками;
- `w_pad` – расстояние по горизонтали между графиками;
- `rect` – параметры прямоугольника в относительных координатах, в который будут вписаны все графики.

В примере 1.2 демонстрируется применение дополнительной настройки графика функции.

Пример 1.2. Построение графиков функций $\sin(x)$ и $\cos(x)$.

```
In[3]:
import numpy as np
import matplotlib.pyplot as plt

# Задаем пропорции рисунка
fig = plt.figure(figsize=(15,5))
x = np.arange(0, 3*np.pi, 0.3)  # Набор значений абсцисс

# Создаем на рисунке первый график
ax1 = fig.add_subplot(121)
# Строим кривую графика функции на первом графике
ax1.plot(x, np.sin(x), marker='*',
         markersize=14, color='r')

# Добавляем подписи осей
ax1.set_xlabel('x', fontsize=20)
ax1.set_ylabel('y', fontsize=20)
# Применяем параметры к обеим осям (axis='both') и
# задаем размер (labelsize) подписи делений оси
ax1.tick_params(axis='both', labelsize=20)

# Создаем на рисунке второй график
ax2 = fig.add_subplot(122)
ax2.plot(x, np.cos(x), marker='o',
```

```
        markersize=14, linestyle='--', color='b')
ax2.set_xlabel('x', fontsize=20)
ax2.set_ylabel('y', fontsize=20)
ax2.tick_params(axis='both', labelsize=20)
fig.tight_layout(w_pad=1)
```

Результат работы этого кода представлен на рисунке 1.2.

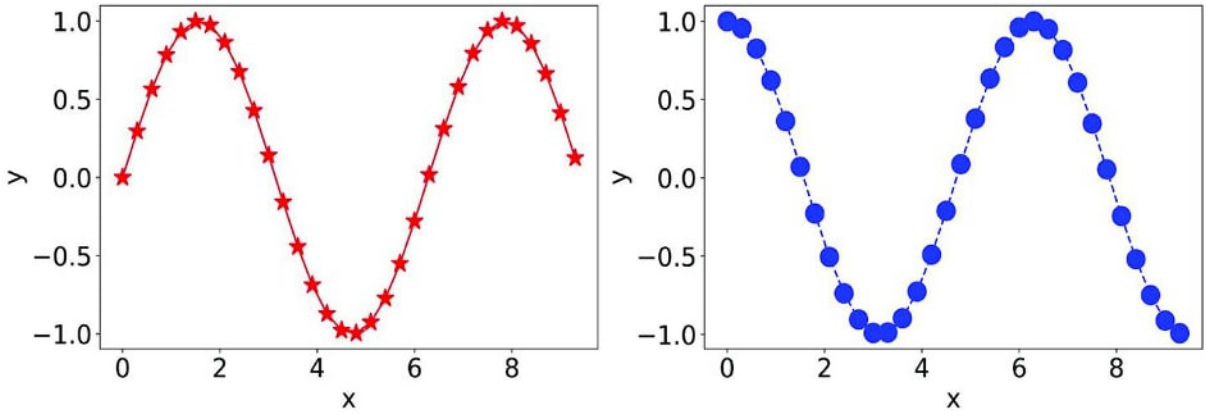


Рисунок 1.2 – Результат работы программы из примера 1.2

В примере 1.2 мы создаем рисунок (`plt.figure`), задаем его размер (`figsize`), создаем графики (`fig.add_subplot`), и получаем тем самым доступ ко всем их элементам. В дальнейшем чаще всего будем использовать именно этот подход к построению графиков.

Если выражение, представляющее функцию, довольно громоздкое, а к функции приходятся часто обращаться, данную функцию в Python можно задать один раз как некоторую новую функцию и в дальнейшем в коде вызывать её.

Рассмотрим способ создания собственной функции.

Особенностью синтаксиса языка Python является применение форматирования в виде отступов строк для объединения их в один блок, вместо, например, ключевых слов begin и end, обозначающих начало и конец блока. Такое форматирование применяется для задания функций, циклов, условий и т.д. Вход в управляемый блок кода указывается с помощью двоеточия. После этого каждая строка кода, которая входит в управляемый блок, должна быть выделена с помощью отступа (четыре пробела).

Функция в Python задается с помощью инструкции `def`:

```
def название_функции(аргументы):
    return значение_функции
```

Результатом работы функции будет выражение, стоящее после инструкции `return` (пример 1.3).

Пример 1.3. Задание функции, которая возводит аргумент в степень 2.

```
In[4]:
def kvadrat(limit):
    return limit**2

print(kvadrat(10))

Out[4]:
100
```

1.2 Практическая работа

Целью работы является ознакомление с базовым синтаксисом языка Python, основными методами задания функций, элементарными математическими операциями и командами в среде Python и пакете NumPy, а также способами графического представления данных в пакете Matplotlib.

1.2.1 Пример выполнения работы

Построим для примера два графика одной функции: заданной с помощью команды `def` и записанной непосредственно в функции `plot()`. Проверим, что графики не зависят от способа задания функции.

Начинаем с импорта необходимых библиотек и модулей:

```
import numpy as np
import matplotlib.pyplot as plt
```

Далее создаем рисунок для размещения на нем графиков:

```
fig = plt.figure(figsize=(15,5))
# Задаем заголовок рисунка, y - положение надписи
fig.suptitle('Построение графика функции',
             y=1, fontsize=18)
```

Создаем набор значений аргумента:

```
x = np.arange(0, 5*np.pi, 0.5)
```

Задаем функцию:

```
def func(x):
    return np.sin(x)*x
```

Создаем первый график:

```
ax1 = fig.add_subplot(121)
```

Далее строим сам график функции, настраиваем его параметры и параметры первого графика:

```
ax1.plot(x, func(x),
         marker='*',
         markersize=12,
         color='r',
         label="Задание функции через def")
ax1.set_xlabel('x', fontsize=16)
ax1.set_ylabel('f(x)', fontsize=16)
ax1.tick_params(axis='both', labelsize=16)

# Деления оси x поворачиваем на 45 градусов
ax1.tick_params(axis='x', labelrotation=45)

# Задаем легенду графика
ax1.legend(fontsize=16)
```

```
# Добавляем линии основной сетки на график
ax1.grid(which='major', color='gray', linestyle=':')
```

То же самое делаем со вторым графиком:

```
ax2 = fig.add_subplot(122)
ax2.plot(x, np.sin(x)*x,
         marker='o',
         markersize=12,
         linestyle='--',
         color='b',
         label="Запись функции в plot()")
ax2.set_xlabel('x', fontsize=18)
ax2.set_ylabel('f(x)', fontsize=18)
ax2.tick_params(axis='both', labelsize=18)

ax2.legend(fontsize=16)

ax2.grid(which='major', color='gray', linestyle='--')
```

Задаем расстояние между графиками (subplot):

```
fig.tight_layout(w_pad=5)
```

Результат работы программы представлен на рисунке 1.3.

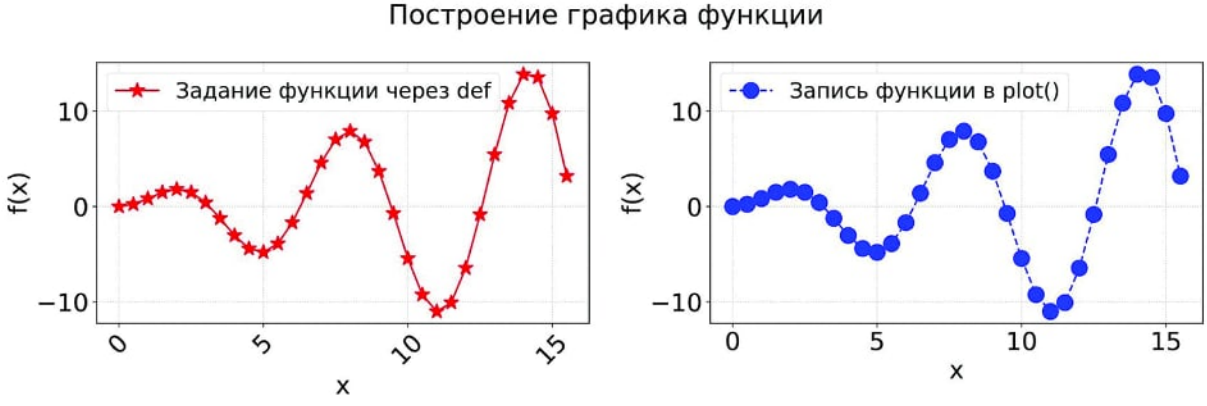


Рисунок 1.3 – Результат работы программы

Как видно из рисунка 1.3 оба способа записи функции дают одинаковый результат.

1.2.2 Задания для самостоятельной работы

В данной работе требуется провести компьютерное моделирование математической функции средствами языка Python. Для этого нужно построить и сравнить графики функции, заданной с помощью команды `def`, и путём записи математического выражения функции непосредственно в метод `plot()`.

В работе должны быть решены следующие задачи:

а) Изучить методы построения основных типов графиков в Matplotlib и методы их персонализации (цвета, маркеры и т.д.).

б) Сформулировать словесный алгоритм программы.

в) Построить блок-схему программы.

г) Написать код программы.

д) Для графиков применить дополнительные возможности визуализации: разные цвета и маркеры для графиков, разные размеры шрифтов, подписей осей и т.д. Оба графика расположить на одном рисунке, его фон сделать каким-либо цветом (не белым). На графиках должны быть легенды, весь рисунок и каждый график должны иметь название.

е) Графики расположить двумя способами: рядом друг с другом и один под другим.

ж) Сделать выводы и составить отчет.

Индивидуальные варианты задания:

1. $f(x) = 5\sin(5x) \cdot x^2 + x^3$, интервал $[0, 10\pi]$.
2. $f(x) = 5\cos(3x) \cdot x^2 + x^2$, интервал $[0, 15\pi]$.
3. $f(x) = 5\sin(5x) \cdot x + x^3 \cos(x)$, интервал $[0, 20\pi]$.
4. $f(x) = 2\sin(10x) + x\cos(x)$, интервал $[0, 10\pi]$.
5. $f(x) = 5x\sin(5x) + 10x\cos(x)$, интервал $[0, 15\pi]$.
6. $f(x) = x^2 \sin(2x) + 5x\cos(x^2)$, интервал $[0, 6\pi]$.
7. $f(x) = 5x^2 \sin(7x) - 5x\cos(x)$, интервал $[0, 10\pi]$.
8. $f(x) = 5x^2 \cos(5x) - 5x^3 \cos(2x)$, интервал $[0, 8\pi]$.
9. $f(x) = 5x^2 \cos(5x) - 5x^3 \sin(3x)$, интервал $[0, 5\pi]$.

10. $f(x) = \cos(5x^2) - 5x^2 \cos(x)$, интервал $[0, 5\pi]$.
11. $f(x) = 5x \sin(5x^2) + x^2 \sin(3x)$, интервал $[0, 5\pi]$.
12. $f(x) = 2x^2 \exp(5x) - 3x^3 \cos(2x)$, интервал $[0, 2\pi]$.
13. $f(x) = 5x^2 \cos(5x) - 5\exp(2x)$, интервал $[0, 2\pi]$.
14. $f(x) = 5x^2 \cos(x)\cos(2x)$, интервал $[0, 3\pi]$.
15. $f(x) = x^2 + \sin(x)\sin(2x)$, интервал $[0, 3\pi]$.

1.3 Контрольные вопросы

1. Что такое функция?
2. Как задается функция в Python?
3. Что означает `return` в процедуре объявления функции?
4. В чем особенность форматирования блоков кода в Python?
5. Для чего предназначена библиотека Matplotlib?
6. Для чего предназначен пакет NumPy?
7. Как вызывается функция из пакета?
8. Как построить несколько графиков с помощью библиотеки Matplotlib?
9. Какая функция используется для построения двухмерного графика с помощью библиотеки Matplotlib?
10. Как добавить подписи к осям в Matplotlib?
11. Как добавить надпись (текст) на график в Matplotlib?
12. Методы построения графиков функций в Matplotlib.

Тема №2. Численное интегрирование функции

2.1 Краткие теоретические сведения

Если подынтегральная функция задана в аналитическом виде, определенный интеграл от нее можно вычислить по формуле Ньютона-Лейбница

$$\int_a^b f(x)dx = F(b) - F(a), \qquad (2.1)$$

где $F(x)$ – первообразная функции $f(x)$.

На практике формулой (2.1) часто не удается воспользоваться по двум основным причинам: функция имеет сложный вид, или не удается аналитически вычислить ее первообразную; подынтегральная функция задана таблично. В этих случаях используют приближенные методы интегрирования.

Приближенные методы вычисления определенных интегралов основаны на замене подынтегральной функции $f(x)$ аппроксимирующей функцией $\varphi(x)$, для которой можно легко записать первообразную.

Чтобы уменьшить погрешность интервал интегрирования $[a,b]$ разбивают на n отрезков и на каждом из полученных отрезков заменяют подынтегральную функцию аппроксимирующей функцией $\varphi_i(x)$. Тогда значение интеграла на интервале $[a,b]$ приближенно равно сумме интегралов от функций $\varphi_i(x)$, взятых в пределах от x_{i-1} до x_i:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} \varphi_i(x)dx. \qquad (2.2)$$

Различные методы численного интегрирования определяются разными способами выбора функций $\varphi_i(x)$ [4].

В методе прямоугольников подынтегральную функцию $f(x)$ на отрезке интегрирования $[x_{i-1}, x_i]$ заменяют константой $f(x) \approx \varphi_i(x) = c_i$. Подставляя $\varphi_i(x) = c_i$ в формулу (2.2) и выполняя интегрирование, получаем

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} c_i \int_{x_{i-1}}^{x_i} dx = \sum_{i=1}^{n} c_i (x_i - x_{i-1}) = \sum_{i=1}^{n} c_i h_i \ . \qquad (2.3)$$

Таким образом, приближенное значение интеграла равно сумме площадей прямоугольников, одна сторона которых равна длине отрезка интегрирования $h_i = x_i - x_{i-1}$, а другая константе c_i.

В зависимости от способа выбора значения c_i метод прямоугольников разделяется на варианты: метод левых прямоугольников, если c_i равно значению функции в точке x_{i-1}; метод правых прямоугольников, если c_i равно значению функции в точке x_i; метод средних прямоугольников, в котором c_i равно значению функции в середине отрезка $[x_{i-1}, x_i]$ (рис. 2.1).

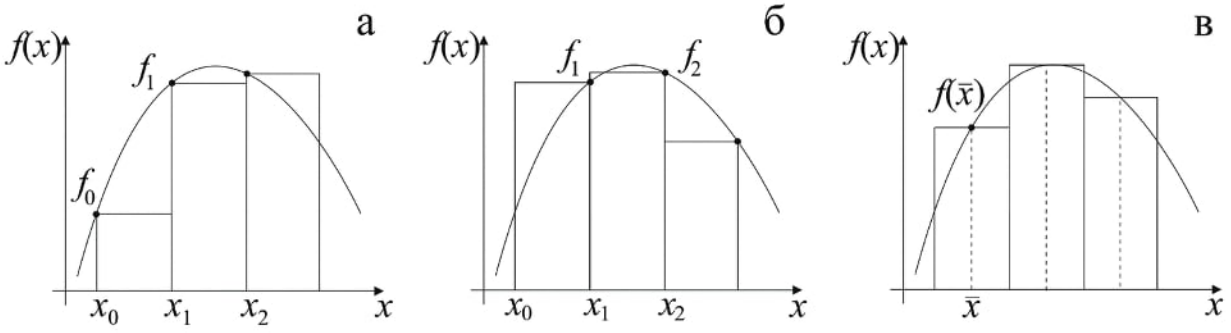


Рисунок 2.1 – Варианты метода прямоугольников: метод левых (а); метод правых (б); метод средних (в)

Более точным методом является метод трапеций. В этом методе интеграл на каждом отрезке $[x_{i-1}, x_i]$ считается приближенно равным площади прямоугольной трапеции:

Как видно из рисунка 1.3 оба способа записи функции дают одинаковый результат.

1.2.2 Задания для самостоятельной работы

В данной работе требуется провести компьютерное моделирование математической функции средствами языка Python. Для этого нужно построить и сравнить графики функции, заданной с помощью команды `def`, и путём записи математического выражения функции непосредственно в метод `plot()`.

В работе должны быть решены следующие задачи:

а) Изучить методы построения основных типов графиков в Matplotlib и методы их персонализации (цвета, маркеры и т.д.).

б) Сформулировать словесный алгоритм программы.

в) Построить блок-схему программы.

г) Написать код программы.

д) Для графиков применить дополнительные возможности визуализации: разные цвета и маркеры для графиков, разные размеры шрифтов, подписей осей и т.д. Оба графика расположить на одном рисунке, его фон сделать каким-либо цветом (не белым). На графиках должны быть легенды, весь рисунок и каждый график должны иметь название.

е) Графики расположить двумя способами: рядом друг с другом и один под другим.

ж) Сделать выводы и составить отчет.

Индивидуальные варианты задания:

1. $f(x) = 5\sin(5x) \cdot x^2 + x^3$, интервал $[0, 10\pi]$.
2. $f(x) = 5\cos(3x) \cdot x^2 + x^2$, интервал $[0, 15\pi]$.
3. $f(x) = 5\sin(5x) \cdot x + x^3 \cos(x)$, интервал $[0, 20\pi]$.
4. $f(x) = 2\sin(10x) + x\cos(x)$, интервал $[0, 10\pi]$.
5. $f(x) = 5x\sin(5x) + 10x\cos(x)$, интервал $[0, 15\pi]$.
6. $f(x) = x^2 \sin(2x) + 5x\cos(x^2)$, интервал $[0, 6\pi]$.
7. $f(x) = 5x^2 \sin(7x) - 5x\cos(x)$, интервал $[0, 10\pi]$.
8. $f(x) = 5x^2 \cos(5x) - 5x^3 \cos(2x)$, интервал $[0, 8\pi]$.
9. $f(x) = 5x^2 \cos(5x) - 5x^3 \sin(3x)$, интервал $[0, 5\pi]$.

Таблица 2.1 – Условные и логические операторы в Python

Оператор	Описание
a < b	a строго меньше b
a > b	a строго больше b
a = = b	a равно b
a != b	a не равно b
a <= b	a меньше или равно b
a >= b	a больше или равно b
условие_1 and условие_2	Верно и условие_1 и условие_2
условие_1 or условие_2	Верно или условие_1, или условие_2, или оба
not условие	Если условие True, то получим результат False

В языке Python есть два типа циклов: цикл `for` и `while` [5].

Цикл `while` позволяет выполнить одну и ту же последовательность действий, пока верно проверяемое условие:

```
while условие:
    блок инструкций
```

В цикле `for` указывается переменная и множество значений, которые она будет последовательно принимать на каждом шаге выполнения цикла:

```
for условие:
    блок инструкций
```

Множество значений переменной цикла может быть задано по-разному: списком, строкой, массивом и т.д.

В цикле могут быть другие циклы. Такие циклы называются вложенными.

Работу циклов `while` и `for` демонстрируют примеры 2.1 и 2.2.

Пример 2.1(а). Работа цикла `while`.

```
In[1]:
i = 1
while i <= 10:
    print(i)
    i = i*4
else:
    print('Конец цикла')

Out[1]:
1
4
Конец цикла
```

Пример 2.1(б). Работа цикла `for`.

```
In[2]:
for i in range(1,6,2):
    print(i)

Out[2]:
1
3
5
```

Пример 2.2(а). Формирование списка с помощью цикла `for`.

```
In[3]:
my_list = []
for i in range(0, 5):
    # Добавляем в конец списка значение i
    my_list.append(i)
print(my_list)

Out[3]:
    [0, 1, 2, 3, 4]
```

Пример 2.2(б). Формирование списка с помощью цикла `for` (цикл внутри списка).

```
In[4]:
my_list = [i for i in range(0, 5)]
print(my_list)

Out[4]:
[0, 1, 2, 3, 4]
```

2.2 Практическая работа

Цель работы: познакомиться с условными операторами и операторами циклов на языке Python, а также методами численного интегрирования.

2.2.1 Пример выполнения работы

В качестве функции, интеграл которой мы будем вычислять, рассмотрим $x^2\cos(x)$ на интервале [0, 12].

Возьмем аналитически интеграл от этой функции, и в дальнейшем с ним будем сравнивать численный расчет:

$$\int_a^b x^2\cos(x)dx = \left(x^2\sin(x) + 2x\cos(x) - 2\sin(x)\right)\Big|_a^b. \qquad (2.5)$$

Для численного интегрирования функции на отрезке $[a, b]$ используем метод правых прямоугольников:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{N} f(x_i)\Delta x = \sum_{i=1}^{N} f(x_0 + i\Delta x)\Delta x, \qquad (2.6)$$

где $\Delta x = (b-a)/N$ – шаг интегрирования, $x_0 = a$ – левая граница интервала интегрирования, $x_N = b$ – правая граница.

Суммирование в (2.6) будем осуществлять в цикле, в котором к посчитанной сумме на предыдущей итерации цикла будет добавляться новое значение при каждой новой итерации.

Для начала загружаем необходимые нам библиотеки:

```
import numpy as np
import matplotlib.pyplot as plt
```

Далее задаем две функции: `fun` – функция, которую надо численно проинтегрировать и `real_integral` – функция аналитически посчитанного интеграла (2.5):

```
def fun(x):
    return x**2*np.cos(x)

def real_integral(a,b):
    real_integral = b**2*np.sin(b) + 2*b*np.cos(b)\
                    - 2*np.sin(b) - a**2*np.sin(a)\
                    - 2*a*np.cos(a) + 2*np.sin(a)
    return real_integral
```

Далее вводим параметры задачи:

```
A = 0              # Начало отрезка интегрирования
B = 12             # Конец отрезка интегрирования
N = 500            # Число точек
DELTA_X =(B-A)/N   # Шаг интегрирования
```

Теперь перейдем к вычислению интеграла методом правых прямоугольников по формуле (2.6):

```
# Начальное значение суммы площадей прямоугольников
s = 0
# Цикл суммирования площадей прямоугольников
for i in range(1, N+1):
    s = s + fun(A + i*DELTA_X)*DELTA_X  # Сумма площадей
```

Функция `range(k,m)` создает набор значений от `k` (включительно) до `m` (не включительно) с шагом 1. Т.к. нам для суммы нужно включить точку $x_N = b$ (правая сторона последнего прямоугольника), то в `range(k,m)` правую границу увеличиваем на величину шага (`m = N + 1`).

Далее выводим на экран значения численно посчитанного интеграла и точное значение.

Окончательно получим следующий код и результат:

```
In[5]:
import numpy as np

def fun(x):
    return x**2*np.cos(x)

def real_integral(a,b):
    real_integral = b**2*np.sin(b) + 2*b*np.cos(b)\
                    - 2*np.sin(b) - a**2*np.sin(a)\
                    - 2*a*np.cos(a) + 2*np.sin(a)
    return real_integral

A = 0              # Начало отрезка интегрирования
B = 12             # Конец отрезка интегрирования
N = 500            # Число точек
DELTA_X =(B-A)/N   # Шаг интегрирования
s = 0
for i in range(1, N+1):
    s = s + fun(A + i*DELTA_X)*DELTA_X

print(s)
print(real_integral(A,B))

Out[5]:
-54.47799873190353
-55.94085934648196
```

Как видно, результаты численно посчитанного интеграла и точного расчета близки.

Далее предоставим пользователю выбор: вывести значение численно посчитанного интеграла и значение погрешности расчета, или график погрешности данного метода в зависимости от количества интервалов разбиения N.

Относительная погрешность вычисляется как

$$\delta = \left|(I_{num} - I_{real})/I_{real}\right|, \qquad (2.7)$$

где I_{num} – значение численного расчета интеграла по формуле (2.6), I_{real} – точное значение интеграла по формуле (2.5).

Выбор дальнейшего хода программы пользователем осуществляется путем ввода им с клавиатуры ответа (`answer`) на запрос программы. Для этого используем оператор условия `if` и `elif`.

Ввод значений с клавиатуры осуществляется с помощью функции `input()`.

Для расчета погрешности создадим функцию вычисления относительной погрешности `pogresh(a,b,n)`, где `a` – левая граница интервала интегрирования, `b` – его правая граница, `n` – количество интервалов разбиения. В этой функции используем численный расчет интеграла, записанный в виде функции `num_integral()`:

```
def num_integral(a, b, n):
    DELTA_X = (b - a)/n
    s = 0
    for i in range(1, n+1):
        s = s + fun(a + i*DELTA_X)*DELTA_X
    return s

def pogresh(a, b, n):
    pogr = abs((num_integral(a, b, n)
                - real_integral(a,b))/real_integral(a,b))
    return pogr
```

Окончательно получим следующий код:

```
In[6]:
import numpy as np
import matplotlib.pyplot as plt

def fun(x):
    return x**2*np.cos(x)

def real_integral(a,b):
    real_integral = b**2*np.sin(b) + 2*b*np.cos(b)\
                    - 2*np.sin(b) - a**2*np.sin(a)\
                    - 2*a*np.cos(a) + 2*np.sin(a)
    return real_integral

def num_integral(a, b, n):
    DELTA_X = (b - a)/n
    s = 0
    for i in range(1, n+1):
        s = s + fun(a + i*DELTA_X)*DELTA_X
    return s

def pogresh(a, b, n):
    pogr = abs((num_integral(a, b, n)
                - real_integral(a,b))/real_integral(a,b))
    return pogr

answer = input('Введите 1, если требуется график'
           + ' погрешности, 0 – если значение интеграла:')

if answer == '1':
    a = float(input('Введите начало интервала A ='))
    b = float(input('Введите конец интервала B ='))
    n = int(input('Введите максимальное количество'
                  + 'интервалов N ='))
    fig = plt.figure(figsize = (10,5))
    ax = fig.add_subplot()
    x = range(2, n + 1)
```

```
    y = [pogresh(a, b, i) for i in x]
    ax.plot(x, y)
    ax.set_xlabel('N', fontsize=20)
    ax.set_ylabel('Отн. погрешность', fontsize=20)
    ax.tick_params(axis='both', labelsize=20)

elif answer == '0':
    a = float(input('Введите начало интервала A ='))
    b = float(input('Введите конец интервала B ='))
    n = int(input('Введите количество интервалов N ='))
    print('Численное значение интеграла равно:',
          num_integral(a, b, n))
    print('Погрешность вычисления равна:',
          pogresh(a, b, n))

else:
    print('Неправильный ввод')
```

Результат работы этого кода для случая вывода графика погрешности представлен на рисунке 2.2.

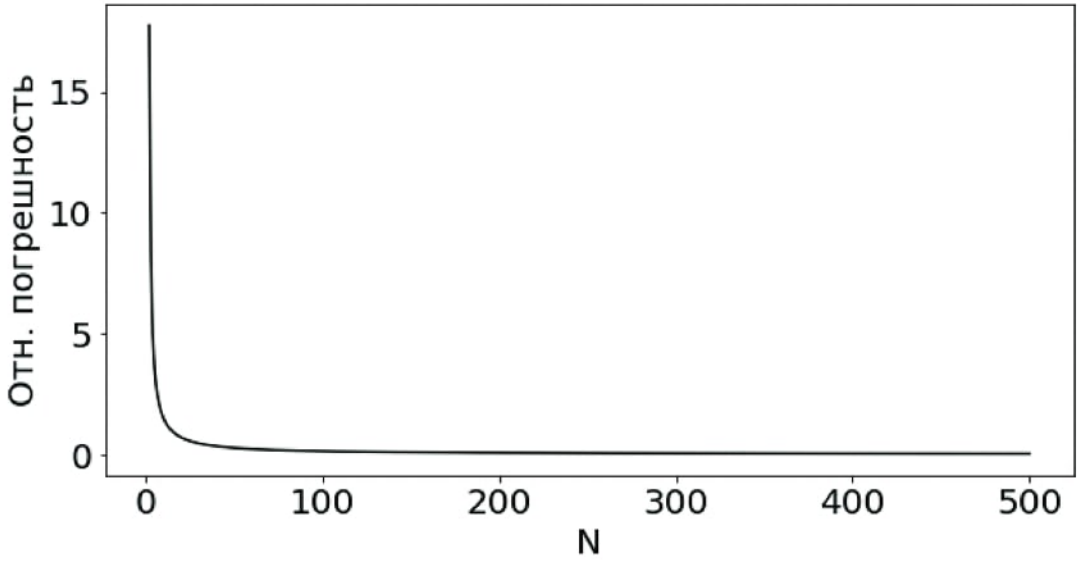


Рисунок 2.2 – График погрешности численного интегрирования методом правых прямоугольников

Как видно из рисунка 2.2 с ростом количества интервалов, на которые разбивается отрезок интегрирования (т.е. с уменьшением шага интегрирования) относительная погрешность численного вычисления интеграла по формуле (2.6) стремится к 0.

2.2.2 Задания для самостоятельной работы

Работа включает два задания.

В первом задании требуется написать код программы, которая вычисляет численно интеграл функции тремя методами: методом левых прямоугольников, методом средних прямоугольников и методом трапеций.

Программа должна спросить пользователя, каким методом вычислить интеграл и в зависимости от ответа пользователя вывести значение численного расчета интеграла по одному из методов.

В ходе выполнения первого задания требуется:

а) сформулировать словесный алгоритм программы;

б) построить блок-схему программы;

в) написать код программы;

г) провести отладку кода;

д) проверить результат.

Во втором задании требуется провести исследование погрешности каждого из методов численного интегрирования, реализованных в первом задании: сравнить погрешности и определить наиболее точный из методов.

На основе полученных результатов нужно определить значение количества интервалов, начиная с которого все три метода дают погрешность меньше 0.1%.

Индивидуальные варианты задания:

1. $x^3 \cos(x) + 2$ на интервале [0, 10].
2. $x^2 \sin(x) - 1$ на интервале [0, 10].
3. $x \cos(x) + x$ на интервале [0, 10].
4. $x^2 \cos(x) + 2$ на интервале [0, 10].
5. $x \sin(x) + 2x$ на интервале [0, 10].
6. $x^2 + 2x \cos(x)$ на интервале [0, 10].

7. $x + 2x\cos(x)$ на интервале [0, 10].
8. $2 + 2x^3\cos(x)$ на интервале [0, 10].
9. $5 + 2x\sin(x)$ на интервале [0, 10].
10. $1 + 2x\sin(x)$ на интервале [0, 10].
11. $e^x + 2x\cos(x)$ на интервале [0, 5].
12. $e^x + 2\sin(x)$ на интервале [0, 5].
13. $e^x + \cos(x)$ на интервале [0, 5].
14. $e^x + \sin(x) + 1$ на интервале [0, 5].
15. $2e^x + 2\sin(x)$ на интервале [0, 5].

2.3 Контрольные вопросы

1. Что такое цикл?
2. Как задается цикл `while` в Python?
3. Как задается цикл `for` в Python?
4. Что такое вложенные циклы и как они работают?
5. Как задается и работает условный оператор в Python?
6. Что такое аппроксимация?
7. Виды аппроксимации функции.
8. Принцип численного интегрирования, его математическая модель и графическое пояснение.
9. Численное интегрирование методом прямоугольников: математическая модель и графическое пояснение.
10. Численное интегрирование методом трапеций: математическая модель и графическое пояснение.
11. В чем заключается погрешность численного интегрирования?

Тема №3. Статистический анализ с Pandas

3.1 Краткие теоретические сведения

Для построения модели какого-либо объекта или явления над ним производятся наблюдения, в результате которых исследователь получает набор данных, отражающих величину измеряемой характеристики этого объекта.

Любое измерение обязательно сопровождается погрешностью. При единичном измерении погрешность может принимать произвольное значение. Поэтому по одному измерению невозможно судить о величине измеряемой характеристики объекта. Для выяснения величины погрешности и определения с достаточной достоверностью истинного значения измеряемой характеристики производится большое количество измерений.

Для обработки результатов многократных измерений каких-либо величин, в том числе изменяющихся во времени, используются методы математической статистики.

Одной из задач статистического анализа данных является определение параметров их функции распределения. Решение этой задачи позволяет определить доверительный интервал, в котором находится истинное значение измеренной величины.

Параметры функции распределения делятся на два типа: центральные характеристики и характеристики рассеяния [6].

Центральные (средние) характеристики показывают положение максимума (или максимумов) распределения, или, другими словами, наиболее типичные значения анализируемых данных. Примером центральных характеристик являются: математическое ожидание, мода, медиана.

Математическим ожиданием (средним арифметическим) называется сумма произведений всех уникальных элементов x_i на соответствующие им частоты n_i, деленная на сумму частот

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{m} x_i n_i \,, \tag{3.1}$$

где m – количество уникальных элементов выборки, N – общее количество элементов выборки.

Медиана – это такое значение, что половина элементов вариационного ряда меньше медианного значения, а половина больше. Т.о. медиана – это значение, которое делит некоторый упорядоченный набор данных на две равные половины.

Мода – значение данных, встречающееся наибольше число раз.

Характеристики рассеяния (разброса) показывают величину разброса данных и форму графика функции распределения. Примеры характеристик разброса: квантили и квартили, дисперсия, размах, интерквартильный размах, стандартное отклонение (среднеквадратическое отклонение).

Дисперсия – это средний квадрат отклонения значений данных от их среднего значения:

$$D = \sigma^2 = \frac{1}{N}\sum_{i=1}^{m}(x_i - \bar{x})^2 n_i \,, \tag{3.2}$$

где σ – стандартное отклонение (среднеквадратическое отклонение).

Размах – разность между максимальным и минимальным значениями ряда:

$$R(X) = x_{\max} - x_{\min} \,. \tag{3.3}$$

Квантиль – это значение, ниже которого лежит определённое число наблюдений, соответствующих выбранной частоте (вероятности).

Три квантиля: 0.25-й, 0.5-й и 0.75-й называются 1-м (нижним, Q_1), 2-м (средним, Q_2) и 3-м (верхним, Q_3) квартилями соответственно.

Интерквартильный размах (IQR) – разность между верхним и нижним квартилями:

$$\mathrm{IQR} = Q_3 - Q_1 \,. \tag{3.4}$$

Квартили могут использоваться для удаления из данных ошибок и выбросов – значений, существенно отличающиеся от

значений данных. Выбросы лежат в интервалах $[Q_1 - 1.5 \times IQR, Q_1 - 3 \times IQR]$ и $[Q_3 + 1.5 \times IQR, Q_3 + 3 \times IQR]$, а грубые ошибки вне интервала $[Q_1 - 3 \times IQR, Q_3 + 3 \times IQR]$.

Одним из пакетов для анализа данных в Python является библиотека Pandas. Эта библиотека отлично подходит для работы с табличными данными. Рассмотрим ее основные возможности.

Библиотека Pandas включает две структуры для работы с данными: `Series` и `DataFrame`.

Структура `Series` представляет собой одномерный объект, который можно представить как таблицу с одной строкой.

`DataFrame` – это двумерная структура, т.е. таблица с множеством строк и столбцов. Стока и столбец объекта `DataFrame` являются одномерными структурами `Series`.

Далее рассмотрим работу со структурой `DataFrame`.

В Pandas объект `DataFrame` создается с помощью следующего конструктора:

```
pandas.DataFrame(data=None, index=None, columns=None, etc.)
```

где `data` – массив, словарь или другой `DataFrame`; `index` – список меток для записей (имена строк таблицы); `columns` – список меток для полей (имена столбцов таблицы).

Структуру `DataFrame` можно создать разными способами.

Создадим объект `DataFrame` (таблицу) с помощью словаря (пример 3.1).

Пример 3.1. Создание таблицы с помощью словаря.

```
In[1]:
import pandas as pd

df = pd.DataFrame({'one': [2, 1, 3, 5],
                   'two': [6, 7, 8, 9],
                   'three': [10, 11, 12, 13]})

print(df)
```

```
Out[1]:
```

	one	two	three
0	2	6	10
1	1	7	11
2	3	8	12
3	5	9	13

Как видно из примера 3.1 по умолчанию строки помечаются числовыми индексами, причем, нумерация начинается с нуля.

Создадим теперь таблицу с заданными маркерами строк и столбцов (пример 3.2).

Пример 3.2. Создание таблицы с заданными метками строк и столбцов с помощью списка.

```
In[2]:
import pandas as pd

df = pd.DataFrame([[1, 2, 3],
                   [10, 20, 30]],
                   index=['s1', 's2'],
                   columns=['c1','c2','c3'])

print(df)
```

```
Out[2]:
```

	c1	c2	c3
s1	1	2	3
s2	10	20	30

Ознакомиться с различными функциями и методами библиотеки Pandas можно с помощью литературы [7, 8].

Основные методы работы с элементами объектов `DataFrame` (считаем, что создан объект `DataFrame` с названием `df`) представлены в таблице 3.1.

Таблица 3.1 – Основные методы работы с элементами объектов `DataFrame`

Метод	Действие
`df['название столбца']`	Выбор столбца по его названию
`df[['a','b','c']]`	Выбор столбцов, названия которых представлены списком ['a', 'b', 'c']
`df[i:n:k]`	Диапазон строк с номера i до номера n-1 с шагом k
`df.loc['метка']`	Выбор строки по метке
`df.loc['i':'j':k]`	Диапазон строк от метки i до метки j включительно с шагом k
`df.loc[['a','b','c']]`	Выбор строк, метки которых представлены списком ['a', 'b', 'c']
`df.loc['строка','столбец']`	Выбор элемента из соответствующих строки и столбца по их названиям
`df.iloc[i]`	Выбор строки с номером i
`df.iloc[i:n:k]`	Диапазон строк с номера i до номера n-1 с шагом k
`df.iloc[[i,j,k]]`	Выбор строк, номера которых представлены списком [i, j, k]
`df.iloc[i, j]`	Выбор элемента из строки и столбца по их номерам

При работе со структурами DataFrame из библиотеки Pandas, как правило, используют два основных способа получения значений элементов. Первый способ основан на использовании (и символьных и численных) меток (или индексов), в этом случае работа ведется через метод .loc[]. Второй способ основан на использовании целых чисел для доступа к данным, он предоставляется через метод .iloc[].

В примере 3.3 показаны разные способы доступа к элементам объекта DataFrame.

Пример 3.3. Вывод конкретного элемента.

```
In[3]:
import pandas as pd

df = pd.DataFrame([[1, 2, 3],
                   [10, 20, 30]],
                   index=['s1', 's2'],
                   columns=['c1','c2','c3'])

print(df['c2'].loc['s2'])
print(df['c2']['s2'])
print(df.loc['s2', 'c2'])
print(df.iloc[1, 1])

Out[3]:
20
20
20
20
```

Важной особенностью работы с объектами DataFrame является их копирование. При обычном присваивании с помощью оператора «=», например, df2=df1, не создается новый объект df2, а создается ссылка на первоначальный df1. Таким образом, любые изменения в df2 приведут к изменениям в df1 (пример 3.4). Чтобы создать новый объект DataFrame нужно воспользоваться функцией copy() (пример 3.5).

Пример 3.4. Создание ссылки на первоначальный объект.

```
In[4]:
import pandas as pd

df1 = pd.DataFrame({'one': [2, 1, 3, 5],
                    'two': [6, 7, 8, 9]})

df2 = df1
# Заменяем элемент 7 на 20 в df2
df2['two'].iloc[1] = 20

print('df1:\n', df1)
```

```
Out[4]:
df1:
```

	one	two
0	2	6
1	1	20
2	3	8
3	5	9

Пример. 3.5. Копирование с созданием нового объекта.

```
In[5]:
import pandas as pd

df1 = pd.DataFrame({'one': [2, 1, 3, 5],
                    'two': [6, 7, 8, 9]})

df2 = df1.copy()
# Заменяем элемент в df2
df2['two'].iloc[1] = 20

print('df1:\n', df1)
print('df2:\n', df2)
```

```
Out[5]:
df1:
```

	one	two
0	2	6
1	1	7
2	3	8
3	5	9

```
df2:
```

	one	two
0	2	6
1	1	20
2	3	8
3	5	9

В библиотеках Python имеется большое число встроенных функций. Так, для сортировки элементов в Pandas используется функция `sort_values()`. Для получения максимального или минимального элемента в строках или столбцах используются методы `max()` или `min()`.

Создадим объект `DataFrame` в виде таблицы, первый столбец которой «one», второй «two» и упорядочим строки по возрастанию элементов первого столбца (пример 3.6).

Пример 3.6. Сортировка элементов.

```
In[6]:
import pandas as pd

df = pd.DataFrame({'one': [2, 1, 3, 5],
                   'two': [6, 7, 8, 9]})

print('Перед сортировкой:')
print(df)
print('После сортировки:')
print(df.sort_values('one', ascending=True))

Out[6]:
```

Перед сортировкой:

	one	two
0	2	6
1	1	7
2	3	8
3	5	9

После сортировки:

	one	two
1	1	7
0	2	6
2	3	8
3	5	9

Найдем сумму элементов столбца, строки, сумму при некотором условии. Для этого применяется функция `sum()` (пример 3.7).

Пример 3.7. Различные варианты суммы элементов с помощью функции `sum()`.

```
In[7]:
import pandas as pd

df = pd.DataFrame({'one': [2, 1, 3, 5],
                   'two': [6, 7, 8, 9]})

summ_column_two = df['two'].sum()  # Сумма в столбце two
summ_line_0 = df.loc[0].sum()  # Сумма в строке 0

print(df)
print('Сумма в столбце two =', summ_column_two,
      '; Сумма в строке 0 =', summ_line_0)

# Сумма при условии, что элемент > 2
s2 = df['one'][df['one'] > 2].sum()
```

```
print('Сумма в столбце one, если элемент > 2, =', s2)
```

Out[7]:

	one	two
0	2	6
1	1	7
2	3	8
3	5	9

```
Сумма в столбце two = 30 ; Сумма в строке 0 = 8
Сумма в столбце one, если элемент > 2, = 8
```

3.2 Практическая работа

Цель работы: познакомиться с основными возможностями библиотеки Pandas, получить базовые практические навыки подготовки данных и их дальнейшего статистического анализа средствами языка Python.

3.2.1 Пример выполнения работы

Проведем предварительный анализ данных, представленных файлом («БД Сотрудники.xlsx») с расширением .xlsx.

Предварительный анализ будет заключаться в решении следующих задач:

а) подготовить данные для анализа;

б) выполнить сортировку элементов;

в) расчет основных статистических показателей;

г) построение графика функции распределения;

д) оценить величину зарплат относительно максимального значения.

Анализ данных будем осуществлять с помощью библиотеки Pandas.

Для начала загрузим данные и выведем часть полученной таблицы для того, чтобы иметь представления о ее структуре:

In[8]:

```
import pandas as pd
```

```
df = pd.read_excel('БД Сотрудники.xlsx')
df.iloc[0:5]
```

Out[8]:

	id	Зарплата	Пол	Возраст	Стаж
0	111	25000	м	35	12
1	112	ывппр	м	NaN	8
2	113	30000	ж	30	7
3	114	26000	м	34	10
4	115	31000	ж	31	11

Выведем общую информацию по данной таблице с помощью метода `info()`:

```
In[9]:
df.info()
```

```
Out[9]:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        32 non-null     object
 1   Зарплата  31 non-null     object
 2   Пол       32 non-null     object
 3   Возраст   31 non-null     object
 4   Стаж      32 non-null     int64
dtypes: int64(1), object(4)
memory usage: 832.0+ bytes
```

По выведенной информации можно получить общие представления о структуре таблицы и данных в ней. Как видно, часть столбцов таблицы, в которых подразумеваются числовые значения, имеют тип `object`. Это означает, что в них присутствуют не числовые значения, например, символы, свидетельствующие об

ошибках ввода данных. Также видно, что часть значений в таблице являются пропусками (значениями NaN).

Перед началом анализа данных их необходимо подготовить: удалить ошибки, дубликаты и пропуски.

Начнем с удаления ошибок. Для этого конвертируем все столбцы, в которых должны быть числа, в числовой формат с помощью метода to_numeric(), а все значения, которые не поддаются конвертации, заменим на NaN:

```
In[10]:
df['id'] = pd.to_numeric(df['id'],
                         errors ='coerce',
                         downcast='integer')

df['Зарплата'] = pd.to_numeric(df['Зарплата'],
                               errors ='coerce',
                               downcast='float')

df['Возраст'] = pd.to_numeric(df['Возраст'],
                              errors ='coerce',
                              downcast='integer')
df.info()

Out[10]:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        31 non-null     float64
 1   Зарплата  30 non-null     float32
 2   Пол       32 non-null     object
 3   Возраст   30 non-null     float64
 4   Стаж      32 non-null     int64
dtypes: float32(1), float64(2), int64(1), object(1)
memory usage: 1.1+ KB
```

Как видно, теперь столбцы с числами приобрели нужный формат, а количество значений NaN увеличилось, что говорит о замене ошибок на NaN.

Выведем строки, в которых есть NaN:

```
In[11]:
df[df.isnull().any(axis=1)]
```

Out[11]:

	id	Зарплата	Пол	Возраст	Стаж
1	112.0	NaN	м	NaN	8
14	133.0	NaN	м	38	12
21	NaN	29000.0	м	NaN	5

Далее удаляем строки с NaN с помощью метода dropna() (удалять строки или столбцы лучше сперва из скопированного DataFrame):

```
In[12]:
df2 = df.copy()
df2 = df2.dropna()
df2.info()

Out[12]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        29 non-null     float64
 1   Зарплата  29 non-null     float32
 2   Пол       29 non-null     object
 3   Возраст   29 non-null     float64
 4   Стаж      29 non-null     int64
dtypes: float32(1), float64(2), int64(1), object(1)
memory usage: 1.1+ KB
```

Как видно, удаление строк с пропусками прошло успешно, теперь во всей таблице их нет.

После удаления всех ошибок и пропусков попробуем снова изменить тип столбцов на более оптимальный:

```
In[13]:
df = df2
df = df.astype({'id': 'int32',
                'Возраст': 'int32',
                'Стаж': 'int32'})

df.info()

Out[13]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29 entries, 0 to 31
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        29 non-null     int32
 1   Зарплата  29 non-null     float32
 2   Пол       29 non-null     object
 3   Возраст   29 non-null     int32
 4   Стаж      29 non-null     int32
dtypes: float32(1), int32(3), object(1)
memory usage: 812.0+ bytes
```

В результате не только тип столбцов изменился на нужный нам, но и произошло уменьшение объема занимаемой таблицей памяти.

Теперь переходим к удалению дубликатов. Для этого сперва выведем все повторяющиеся строки таблицы:

```
In[14]:
df[df.duplicated()]

Out[14]:
```

	id	Зарплата	Пол	Возраст	Стаж
11	113	30000.0	ж	30	7
15	113	30000.0	ж	30	7
16	129	22000.0	ж	26	3

В таблице присутствует три строки, являющиеся копиями других. Удаляем их и проверяем результат:

```
In[15]:
df = df.drop_duplicates()
df[df.duplicated()]
```

```
Out[15]:
```

id	Зарплата	Пол	Возраст	Стаж

Далее построим гистограмму по столбцу зарплат. Для удобства разделим все значения зарплат на 1000 и переименуем столбец «Зарплата» в «Зарплата, тыс. руб.»:

```
In[16]:
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(10,5))
ax = fig.add_subplot()

df.loc[:,'Зарплата'] = df.loc[:,'Зарплата']/1000
df.rename(columns = {'Зарплата':'Зарплата, тыс. руб.'},
          inplace=True)
ax.hist(df['Зарплата, тыс. руб.'])
ax.set_xlabel('Зарплата, тыс. руб.', fontsize=18)
ax.set_ylabel('Количество', fontsize=18)
ax.tick_params(axis ='both', labelsize=18)
```

Результат выполнения кода представлен на рисунке 3.1.

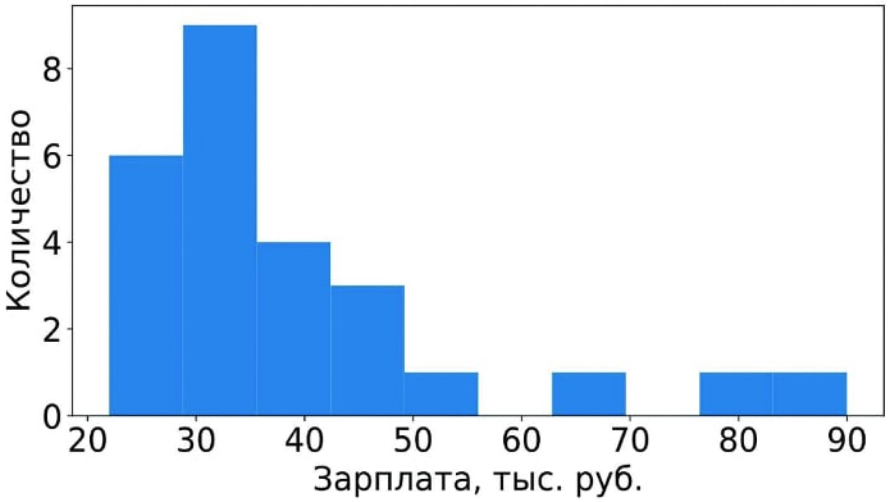


Рисунок 3.1 – Гистограмма распределения зарплат

Выведем теперь основные статистические показатели анализируемой таблицы данных с помощью метода `describe()`:

```
In[17]:
df.describe()
```

Out[17]:

	id	Зарплата, тыс. руб.	Возраст	Стаж
count	29.000	29.000	29.000	29.000
mean	130.655	37.017	36.758	11.344
std	13.039	16.423	7.716	6.309
min	111.000	22.000	26.000	3.000
25%	117.000	29.000	30.000	7.000
50%	132.000	31.000	36.000	10.000
75%	141.000	41.000	41.000	15.000
max	150.000	90.000	56.000	30.000

Добавим теперь к таблице еще один столбец, в котором будет величина зарплат сотрудников относительно максимальной зарплаты. Для того чтобы округлить значение до нужного количества десятичных знаков, используем метод `round()`:

```
In[18]:
df['Зарплата, отн. ед.'] = round(df['Зарплата, тыс. руб.']
                      /df['Зарплата, тыс. руб.'].max(), 2)
df[0:5]
```

Out[18]:

	id	Зарплата, тыс. руб.	Пол	Возраст	Стаж	Зарплата, отн. ед.
0	111	25.0	м	35	12	0.28
2	113	30.0	ж	30	7	0.33
3	114	26.0	м	34	10	0.29
4	115	31.0	ж	31	11	0.34
5	116	30.0	м	33	12	0.33

Отсортируем теперь строки по убыванию зарплаты и заново пронумеруем:

```
In[19]:
df = df.sort_values(by='Зарплата, тыс. руб.',
                    axis=0,
                    ascending=False)
df.reset_index(drop=True, inplace=True)
df[0:5]
```

Out[19]:

	id	Зарплата, тыс. руб.	Пол	Возраст	Стаж	Зарплата, отн. ед.
0	117	90.0	м	45	15	1.00
1	118	80.5	ж	52	25	0.89
2	119	65.0	м	56	30	0.72
3	148	50.0	м	45	20	0.56
4	149	45.0	ж	46	15	0.50

На этом предварительная обработка и анализ данных завершены.

3.2.2 Задания для самостоятельной работы

В ходе выполнения работы необходимо решить следующие задачи в соответствии с условиями индивидуального варианта:

а) подготовить данные для анализа: загрузить данные из файла индивидуального задания, заменить в данных ошибки ввода, удалить дубликаты;

б) выполнить сортировку данных (вручную, написав код, и с помощью функций сортировки из библиотеки Pandas);

в) написать код для вычисления среднего значения и медианы, сравнить результат со значениями, вычисленными с помощью соответствующих функций библиотеки Pandas;

г) построить гистограмму;

д) добавить к таблице новый столбец и строку;

е) сохранить полученную таблицу в новый файл (формат файла по условию индивидуального варианта).

Индивидуальные варианты задания и файлы с данными выдаются преподавателем.

3.3 Контрольные вопросы

1. Что такое математическое ожидание?
2. Что такое медиана?
3. Что такое дисперсия?
4. Что такое центральные (средние) характеристики?
5. Что такое характеристики рассеяния?
6. Что такое мода?
7. Что такое квантиль и квартиль?
8. Какие значения называются выбросами?
9. Какие значения называются грубыми ошибками?
10. Как загрузить из внешнего файла данные с помощью библиотеки Pandas?
11. Что такое объект DataFrame?
12. Как сделать срез данных объекта DataFrame?
13. Какие есть способы обращения к элементам DataFrame?
14. Напишите код, вычисляющий медиану столбца объекта DataFrame.

Тема №4. Сглаживание временного ряда методом скользящего среднего

4.1 Краткие теоретические сведения

При описании данных о развитии того или иного процесса во времени используется понятие временного ряда.

Временной ряд – это совокупность значений какого-либо показателя за несколько последовательных моментов или периодов времени. Например, в экономике временными рядами описывается динамика цен, изменение объема продаж и т.д. Каждое новое значение временного ряда, называемое уровнем, формируется под воздействием различных факторов, которые условно можно разделить на три группы: формирующие тенденцию ряда, формирующие циклические колебания ряда и случайные факторы.

К методам анализа временных рядов относятся:

а) корреляционный анализ – позволяет выявить существенные периодические зависимости и их лаги (задержки);

б) спектральный анализ – позволяет находить периодические и квазипериодические составляющие временного ряда;

в) сглаживание и фильтрация – предназначены для преобразования временного ряда с целью удаления из него высокочастотных и сезонных колебаний;

г) модели авторегрессии и скользящего среднего – для описания и прогнозирования процессов, проявляющих однородные колебания вокруг среднего значения;

д) прогнозирование – позволяет на основе подобранной модели поведения временного ряда предсказывать его значение в будущем.

На графике изменения некоторой величины влияние на нее случайных факторов проявляется в виде «шума». Для того чтобы сгладить колебания величины, применяется сглаживание и фильтрация.

Одним из методов сглаживания является метод скользящего среднего. Если тренд сохраняется, скользящее среднее можно использовать и для прогноза изменения изучаемой величины в

будущем. Скользящие средние бывают различных видов: простые (SMA), экспоненциальные (EMA) и их варианты [9].

Простое скользящее среднее рассчитывается как среднее значение за несколько предыдущих периодов (n):

$$y_{i+1} = \frac{1}{n}\sum_{k=1}^{n} y_{i-k}\ , \tag{4.1}$$

где y_i – значения временного ряда.

Для оценки точности, с которой модель описывает изменение некоторой величины, может применяться коэффициент корреляции.

Если x и y – две случайные величины, коэффициент их корреляции $K(x, y)$ равен

$$K(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}, \tag{4.2}$$

где n – число пар точек x и y.

Для статистически независимых величин $K(x, y) = 0$. Чем ближе модуль коэффициента корреляции к 1, тем более величины связаны друг с другом.

4.2 Практическая работа

Цель работы: познакомиться с основными методами обработки данных временных рядов и их сглаживания методом скользящего среднего с помощью языка Python.

4.2.1 Пример выполнения работы

В данной работе с помощью простого скользящего среднего (SMA) проведем сглаживание временного ряда, представленного значениями котировок пары EUR/USD.

Первый этап анализа заключается в подготовке данных.

Загрузим значения временного ряда из имеющего файла котировок, представляющего собой таблицу данных, и выведем первые три строки:

```
In[1]:
import pandas as pd

ktr = pd.read_csv('EURUSD_190501_191101(day).csv',
                  delimiter=';')

print(ktr.iloc[0:3])
```

Out[1]:

	<DATE>	<TIME>	<OPEN>	<HIGH>	<LOW>	<CLOSE>
0	20190501	00:00:00	1.12142	1.12649	1.1186	1.11949
1	20190502	00:00:00	1.11942	1.12191	1.1169	1.11713
2	20190503	00:00:00	1.11709	1.12050	1.1134	1.12011

Как видно, в таблице представлены значения цены открытия и закрытия, а также максимального и минимального её значения в течение суток и даты в формате «годмесяцдень».

Выведем общую информацию по таблице и проверим наличие пропусков и ошибок:

```
In[2]:
ktr.info()

Out[2]:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116 entries, 0 to 115
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   <DATE>   116 non-null    int64
```

```
 1   <TIME>   116 non-null    object
 2   <OPEN>   116 non-null    float64
 3   <HIGH>   116 non-null    float64
 4   <LOW>    116 non-null    float64
 5   <CLOSE>  116 non-null    float64
dtypes: float64(4), int64(1), object(1)
memory usage: 5.0+ KB
```

Далее удалим ненужный столбец <TIME>. Для более удобного использования значения даты переведем её из формата «годмесяцдень» в формат «год-месяц-день». Для этого применяется функция datetime():

```
In[3]:
# Удаляем столбец <TIME>
ktr = ktr.drop(columns='<TIME>')
# Изменяем формат даты
ktr['<DATE>'] = pd.to_datetime(ktr['<DATE>'],
                               format='%Y%m%d').dt.date
print(ktr.iloc[0:3])
```

Out[3]:

	<DATE>	<OPEN>	<HIGH>	<LOW>	<CLOSE>
0	2019-05-01	1.12142	1.12649	1.1186	1.11949
1	2019-05-02	1.11942	1.12191	1.1169	1.11713
2	2019-05-03	1.11709	1.12050	1.1134	1.12011

Теперь дата состоит из 3-х частей, и к каждой из них можно обращаться отдельно. Например,

```
In[4]:
print('day=', ktr['<DATE>'].iloc[0].day)
print('month=', ktr['<DATE>'].iloc[0].month)
print('year=', ktr['<DATE>'].iloc[0].year)
```

```
Out[4]:
day= 1
month= 5
year= 2019
```

Наша цель заключается в построении графика цены закрытия по дням и скользящего среднего.

Для примера, посчитаем скользящие средние по 3-м и 10-ти предыдущим значениям.

В Pandas для вычисления скользящих средних применяется функция `rolling(n).mean()`, где `n` – период, по которому происходит усреднение.

Посчитанные значения скользящих средних запишем в новые столбцы с названиями «`SMA_3`» и «`SMA_10`», добавляемые к исходной таблице:

```
In[5]:
ktr['SMA_3'] = ktr['<CLOSE>'].rolling(3).mean()
ktr['SMA_10'] = ktr['<CLOSE>'].rolling(10).mean()
```

Для определения точности, с которой скользящие средние описывают поведение исходных данных, воспользуемся коэффициентом корреляции.

В библиотеке NumPy коэффициент корреляции определяется функцией `corrcoef(`y_1`,`y_2`,…,`y_n`)`, которая возвращает матрицу n×n корреляций величин y_1, y_2, …, y_n друг с другом:

```
In[6]:
corr_3 = round(np.corrcoef(ktr['SMA_3'].iloc[2:],
               ktr['<CLOSE>'].iloc[2:])[0,1],3)
corr_10 = round(np.corrcoef(ktr['SMA_10'].iloc[9:],
                ktr['<CLOSE>'].iloc[9:])[0,1],3)
```

Перейдем к построению графиков.

Так как значений дат достаточно много, для удобства по оси Ox будем откладывать не все значения дат, а с шагом 10 дней. В результате получим следующий код:

```
In[7]:
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(15,5))
ax = fig.add_subplot()

ax.plot(ktr['<DATE>'], ktr['<CLOSE>'],
        color ='b',
        label = "close")
ax.plot(ktr['<DATE>'], ktr['SMA_3'],
        color ='r',
        linestyle ='--',
        linewidth = 3,
        label = 'SMA(3), corr=' + str(corr_3))

ax.plot(ktr['<DATE>'], ktr['SMA_10'],
        color ='g',
        linestyle =':',
        linewidth = 4,
        label = 'SMA(10), corr=' + str(corr_10))

# Подписываем ось x с шагом 10 дней
ax.set_xticks([ktr['<DATE>'].iloc[i] for i in
              range(0,len(ktr['<DATE>']),10)])

# Меняем подписи
ax.set_xticklabels([str(ktr['<DATE>'].iloc[i].day) + '-'
                  + str(ktr['<DATE>'].iloc[i].month)
                  for i in range(0,len(ktr['<DATE>']),10)])

ax.tick_params(axis='x', labelrotation = 70)
ax.set_xlabel('Day-Month', fontsize = 18)
ax.set_ylabel('EUR/USD', fontsize = 18)
ax.tick_params(axis='both', labelsize = 18)
ax.legend(fontsize = 14)
ax.grid(which='major', color='gray', linestyle=':')
```

Результат работы кода представлен на рисунке 4.1.

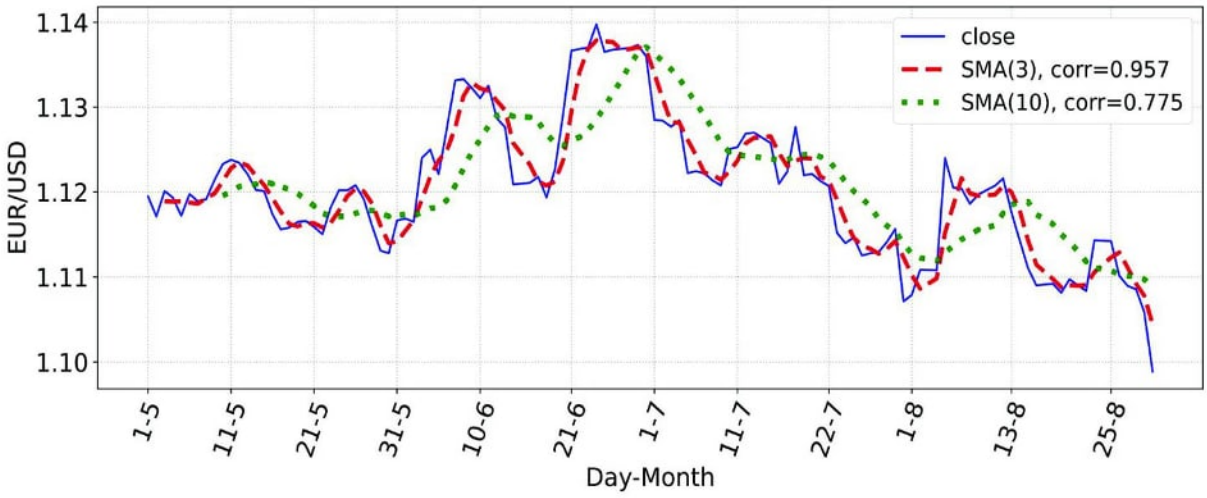


Рисунок 4.1 – График котировок пары EUR/USD и скользящих средних

Как видно из рисунка, скользящее среднее по 3-м периодам имеет гораздо большую точность по сравнению со скользящей по 10-ти периодам.

4.2.2 Задания для самостоятельной работы

В работе требуется построить средствами языка Python и библиотеки Pandas распределения исследуемого временного ряда и провести его сглаживание скользящими средними.

В ходе выполнения работы должны быть решены следующие задачи:

а) сформулировать словесный алгоритм программы;

б) построить блок-схему программы;

в) написать код программы;

г) для графиков применить дополнительные возможности визуализации, доступные в функции `plot()`: разные цвета, маркеры, размеры шрифта;

д) сделать выводы и составить отчет.

Индивидуальные варианты задания и файлы с данными выдаются преподавателем.

```
 1   <TIME>   116 non-null    object
 2   <OPEN>   116 non-null    float64
 3   <HIGH>   116 non-null    float64
 4   <LOW>    116 non-null    float64
 5   <CLOSE>  116 non-null    float64
dtypes: float64(4), int64(1), object(1)
memory usage: 5.0+ KB
```

Далее удалим ненужный столбец <TIME>. Для более удобного использования значения даты переведем её из формата «годмесяцдень» в формат «год-месяц-день». Для этого применяется функция datetime():

```
In[3]:
# Удаляем столбец <TIME>
ktr = ktr.drop(columns='<TIME>')
# Изменяем формат даты
ktr['<DATE>'] = pd.to_datetime(ktr['<DATE>'],
                               format='%Y%m%d').dt.date
print(ktr.iloc[0:3])
```

Out[3]:

	<DATE>	<OPEN>	<HIGH>	<LOW>	<CLOSE>
0	2019-05-01	1.12142	1.12649	1.1186	1.11949
1	2019-05-02	1.11942	1.12191	1.1169	1.11713
2	2019-05-03	1.11709	1.12050	1.1134	1.12011

Теперь дата состоит из 3-х частей, и к каждой из них можно обращаться отдельно. Например,

```
In[4]:
print('day=', ktr['<DATE>'].iloc[0].day)
print('month=', ktr['<DATE>'].iloc[0].month)
print('year=', ktr['<DATE>'].iloc[0].year)
```

Тема №5. Построение регрессионных моделей временного ряда

5.1 Краткие теоретические сведения

Для изучения любых объектов и процессов строятся их упрощенные модели, учитывающие только наиболее важные параметры этих объектов и процессов, а также воздействующей на них внешней среды. Построение упрощенных моделей осуществляется с помощью методов аппроксимации.

Неизвестная функциональная зависимость одной величины, называемой откликом, от других величин, называемых факторами, аппроксимируется известными функциями. При этом возникает задача подбора таких параметров, входящих в эти функции, чтобы замена была наиболее точной. Поиск этих параметров составляет задачу регрессии [6].

Чаще всего используют следующие модели регрессионных функций: линейная, полиномиальная, степенная и логарифмическая.

Рассмотрим полиномиальную модель регрессии и её частный случай – линейную.

Пусть дан упорядоченный набор некоторых значений $x_1, x_2, ..., x_n$ и соответствующие им значения $y_1, y_2, ..., y_n$. Если x_i являются временными интервалами, то двумерный набор данных (x_i, y_i) представляет собой временной ряд.

Анализ временных рядов начинают с их визуализации. Для этого строят распределение $y_i(x_i)$. По виду получившегося распределения выбирают тип регрессии. Пусть, для примера, получено распределение, показанное на рисунке 5.1.

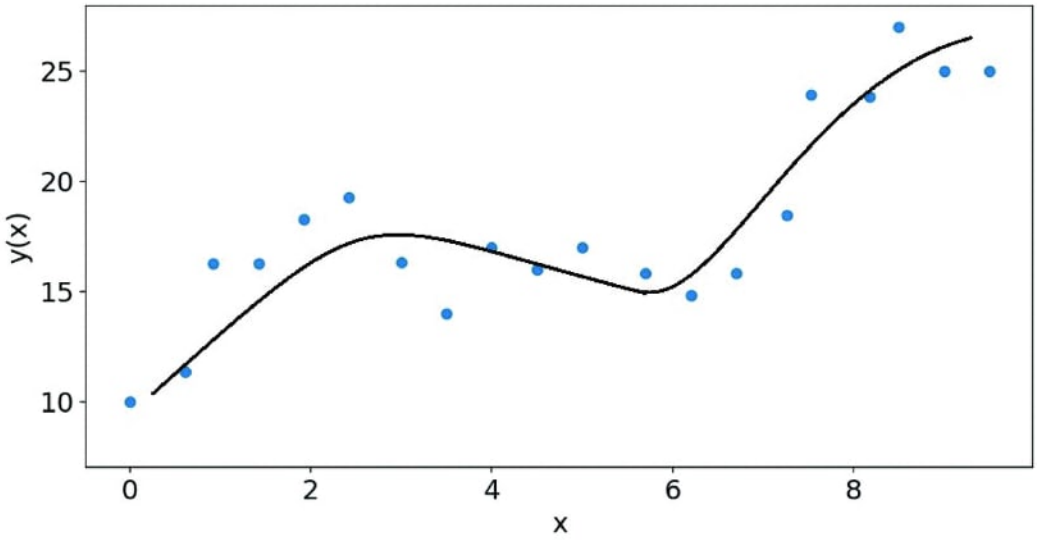


Рисунок 5.1 – Пример временного ряда (точки) и линии полиномиальной регрессии

Как видно из рисунка, линия регрессии описывает неизвестную зависимость $y(x)$, которую можно приближенно заменить уравнением полиномиальной регрессии n-го порядка

$$y(x) \approx a_0 + a_1 x + a_2 x^2 + ... + a_n x^n . \quad (5.1)$$

Если линия регрессии является прямой, то неизвестную зависимость $y(x)$ приближенно заменяют уравнением линейной регрессии

$$y(x) \approx a_1 x + a_2 , \quad (5.2)$$

где a_1 и a_2 – неизвестные коэффициенты регрессии.

Полиномиальная модель регрессии (5.1) может быть сведена к множественной линейной регрессии путем замен $z_i = x^i$ ($i = 0, 1, 2, ..., n$):

$$y(x) \approx \sum_{i=0}^{n} a_i x^i = \sum_{i=0}^{n} a_i z_i . \quad (5.3)$$

Построенная модель регрессии позволяет находить значения неизвестной функции как внутри интервала значений аргумента, на

котором строилась регрессия, так и за его пределами. Т.о. можно получить с точностью регрессии будущие значения неизвестной функции.

Если вид функции регрессии $f(x)$ выбран, оценка неизвестных параметров a_i может быть произведена методом наименьших квадратов. Согласно этому методу неизвестные параметры функции выбираются таким образом, чтобы сумма квадратов отклонений экспериментальных значений y_i от их расчетных (теоретических) значений была минимальной, т.е.

$$\sum_{i=1}^{n}(y_i - f(x_i, a_i))^2 \to \min . \qquad (5.4)$$

В Python линейная регрессия реализована в нескольких пакетах, например, в NumPy и Scikit-learn [10].

Для построения регрессии с помощью библиотеки Scikit-learn используются модули linear_model и preprocessing.

Алгоритм построения регрессии в Scikit-learn следующий.

Создается модель будущей линейной по коэффициентам регрессии как объект класса LinearRegression из модуля linear_model:

```
model = LinearRegression()
```

Далее, к полученной модели регрессии применяется метод `fit()`:

```
model.fit(x,y)
```

Метод `fit(x,y)` вычисляет оптимальные значения коэффициентов регрессии для входных наборов данных: набора x и соответствующего ему набора значений y. При этом набор данных x должен представлять собой вектор-столбец, а y – вектор-строку.

Для парной полиномиальной модели регрессии n-й степени методу `fit(x,y)` в качестве набора данных x требуются для каждого значения x_i еще и значения $x_i^0, x_i^2, x_i^3, ..., x_i^n$, чтобы для

них рассчитать оптимальные значения коэффициентов. Это означает, что исходный вектор-столбец

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_m \end{pmatrix}$$

необходимо преобразовать в матрицу

$$x = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ . & . & . & . & . \\ 1 & x_m & x_m^2 & \dots & x_m^n \end{pmatrix}.$$

Эта операция осуществляется методом `fit_transform()` из класса PolynomialFeatures модуля preprocessing:

```
PolynomialFeatures(degree=n).fit_transform(x)
```

где n – степень полинома. В случае m переменных этот метод вычисляет значения одночленов вида $x_1^{i_1} x_2^{i_2} \cdot \dots \cdot x_m^{i_m}$, при условии $i_1 + i_2 + \dots + i_m \le n$:

$$P_n(x_1, x_2, \dots, x_m) = \sum_{i_1, i_2, \dots, i_m = 0}^{n} x_1^{i_1} x_2^{i_2} \cdot \dots \cdot x_m^{i_m} . \qquad (5.5)$$

5.2 Практическая работа

Цель работы: познакомиться с основными методами линейного по коэффициентам регрессионного анализа временных рядов средствами языка Python и библиотеки Scikit-learn.

5.2.1 Пример выполнения работы

В данной работе требуется построить несколько моделей линейной по коэффициентам регрессии для временного ряда, представленного значениями котировок пары EUR/USD. Из построенных моделей нужно определить наилучшую и сделать с её помощью прогноз значений пары EUR/USD. Для примера мы построим функции регрессии 1-й, 2-й и 7-й степени.

Для работы с данными будем использовать библиотеку Pandas и NumPy, а для построения регрессии библиотеку Scikit-learn.

Загружаем необходимые библиотеки, модули и классы:

```
In[1]:
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

Первый этап работы заключается в подготовке данных.

Загрузим значения временного ряда из имеющего файла котировок EURUSD.csv, представляющего собой таблицу данных, и выведем первые три строки:

```
In[2]:
df = pd.read_csv('EURUSD.csv',delimiter=';')
df['<DATE>']= pd.to_datetime(df['<DATE>'],
                             format='%Y%m%d').dt.date

df.iloc[0:3]
```

```
Out[2]:
```

	<DATE>	<TIME>	<OPEN>	<HIGH>	<LOW>	<CLOSE>
0	2019-05-01	00:00:00	1.1214	1.1264	1.1186	1.1194
1	2019-05-02	00:00:00	1.1194	1.1219	1.1169	1.1171
2	2019-05-03	00:00:00	1.117	1.1205	1.1134	1.1201

Выведем общую информацию по таблице:

```
In[3]:
df.info()

Out[3]:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116 entries, 0 to 115
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   <DATE>   116 non-null    object
 1   <TIME>   116 non-null    object
 2   <OPEN>   116 non-null    float64
 3   <HIGH>   116 non-null    float64
 4   <LOW>    116 non-null    float64
 5   <CLOSE>  116 non-null    float64
dtypes: float64(4), object(2)
memory usage: 4.6+ KB
```

Как видно, пропусков в данных нет. Изменим тип столбцов с float64 на float32:

```
In[4]:
df = df.astype({'<OPEN>': 'float32','<HIGH>': 'float32',
                '<LOW>': 'float32', '<CLOSE>': 'float32'})
df.info()

Out[4]:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116 entries, 0 to 115
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   <DATE>   116 non-null    object
 1   <TIME>   116 non-null    object
 2   <OPEN>   116 non-null    float32
```

```
 3   <HIGH>   116 non-null    float32
 4   <LOW>    116 non-null    float32
 5   <CLOSE>  116 non-null    float32
dtypes: float32(4), object(2)
memory usage: 2.8+ KB
```

Далее переходим к построению графиков функций регрессий.

Построим регрессию для значений цен закрытия (отклик), представленных в столбце `'<CLOSE>'`, и дат (фактор) из столбца `'<DATE>'`.

Создаем необходимые для построения регрессии вектор-столбец x и вектор-строку y. Вместо дат из столбца `'<DATE>'` для регрессии используем их порядковые номера:

```
x = np.arange(0,len(df)).reshape((-1, 1))
y = np.array(df['<CLOSE>'])
```

Функция `reshape(a,b)` переформатирует массив в `'a'` строк и `'b'` столбцов. В записи `reshape(-1,b)` значение -1 означает, что количество строк заранее неизвестно, их столько, сколько получится для формирования b столбцов.

Далее создаем модель будущей линейной по коэффициентам регрессии как объект класса `LinearRegression`:

```
model = LinearRegression()
```

Полиномиальную регрессию сведем к множественной линейной регрессии. Для этого с помощью метода `fit_transform()` из класса `PolynomialFeatures` получим из значений вектора-столбца x матрицу, в i-й строке которой будет находиться набор значений $x_i^0, x_i^2, x_i^3, ..., x_i^n$, где n – степень полинома:

```
reg_two = PolynomialFeatures(degree=2)
reg_seven = PolynomialFeatures(degree=7)
x_two = reg_two.fit_transform(x)
x_seven = reg_seven.fit_transform(x)
```

Наша задача состоит не только в построении моделей регрессии, но и в прогнозировании значений исследуемого временного ряда. Сделаем прогноз на 10 дней. Для этого создаем новый массив значений аргумента, на 10 элементов больше, чем текущий:

```
x_new = np.append(x,np.array([i for i in
                  range(116,126)])).reshape((-1, 1))
```

Метод `np.append(a, b)` добавляет массив `b` в конец массива `a`.

Теперь переходим непосредственно к вычислению регрессий. Начинаем с построения линейной регрессии:

```
model_one = model.fit(x,y)
r2_one = model_one.score(x,y)
y_one_new = model_one.predict(x_new)
```

С помощью метода `fit()` вычисляются оптимальные значения коэффициентов регрессии по значениям данных x и y. Функция `score(x,y)`, примененная к построенной модели регрессии `model`, рассчитывает коэффициент корреляции R^2 между значениями полученными моделью на наборе x и значениями в наборе y. Функция `predict(x_new)`, используя построенную модель регрессии, предсказывает значение в новом массиве аргументов `x_new`.

После этого рассчитываем коэффициенты регрессий 2-го и 7-го порядка:

```
# Квадратичная модель
model_two = model.fit(x_two, y)
r2_two = model_two.score(x_two, y)

# Подготавливаем для регрессии новый набор аргументов x
x_two_new = reg_two.fit_transform(x_new)
y_two_new = model_two.predict(x_two_new) # Прогноз
# Модель регрессии 7-й степени
model_seven = model.fit(x_seven, y)
r2_seven = model_seven.score(x_seven, y)
x_seven_new = reg_seven.fit_transform(x_new)
y_seven_new = model_seven.predict(x_seven_new)
```

Теперь переходим к выводу результатов:

```
ax.plot(x, df['<CLOSE>'],
        color='b',
        linewidth = 2,
        label="close")
ax.plot(x_new, y_one_new,
        color="g",
        linestyle = "-.",
        linewidth = 3,
        label = "n=1, R^2=" + str(round(r2_one,3)))
ax.plot(x_new, y_two_new,
        color="black",
        linestyle = ":",
        linewidth = 4,
        label = "n=2, R^2=" + str(round(r2_two,3)))
ax.plot(x_new, y_seven_new,
        color = "r",
        linestyle = "--",
        linewidth = 4,
        label = "n=7, R^2=" + str(round(r2_seven,3)))
ax.set_xlabel('Day-Month',size = 18)
ax.set_ylabel('EUR/USD', size = 18)

# Подписываем ось x с шагом 7 дней:
ax.set_xticks(range(0,len(x_new),7))

# Создаем список подписей оси x:
list_labels = ['']*len(x_new[::7])

# Заполняем список подписями, кроме прогнозной части:
list_labels[0:len(df.iloc[::7])] = [
        str(df['<DATE>'].iloc[i].day)
        + '-' + str(df['<DATE>'].iloc[i].month) for i
        in range(0,len(df),7)
]
ax.set_xticklabels(list_labels)
```

```
ax.tick_params(axis = 'x',
               labelrotation = 70)
ax.tick_params(axis = 'both',
               labelsize = 16)
ax.legend(fontsize = 16)
ax.grid(which = 'major', color = 'gray', linestyle = ':')
```

Итоговый результат представлен на рисунке 5.2.

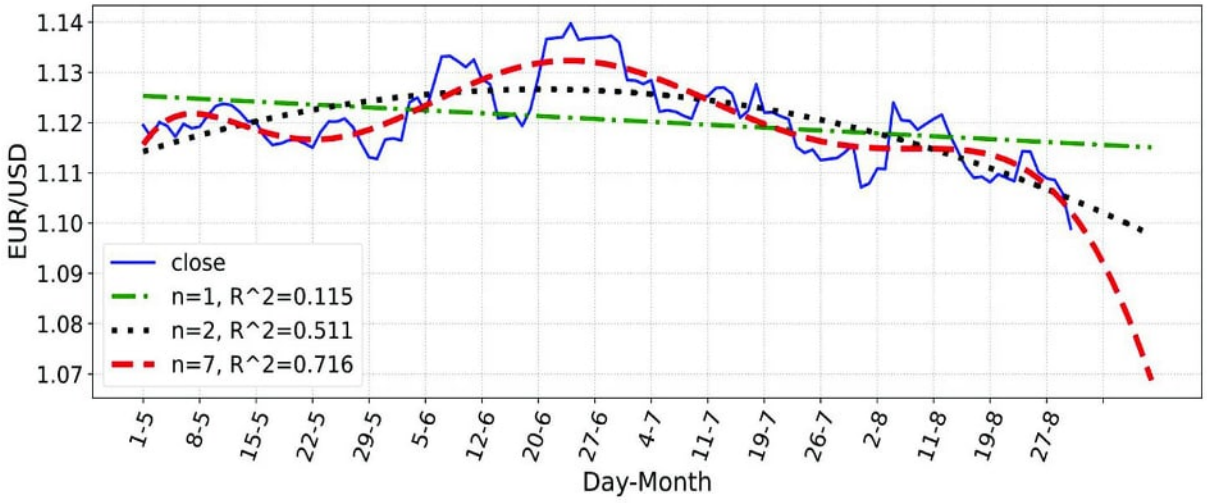


Рисунок 5.2 – Временной ряд котировок пары EUR/USD и линии регрессий

Участки линий регрессии продолжающиеся правее окончания графика цены закрытия являются прогнозом изменения временного ряда. При этом наиболее точной в данном случае является линия регрессии 7-го порядка.

Выведем отдельно предсказываемые значения временного ряда на 10 дней вперед моделью регрессии 7-го порядка:

```
df_new = pd.DataFrame({
               'День, №': [i for i in range(1,11)],
               'Прогноз': list(y_seven_new[len(x):])
})

df_new
```

В результате получим прогноз:

	День, №	Прогноз
0	1	1.100
1	2	1.097
2	3	1.095
3	4	1.092
4	5	1.088
5	6	1.085
6	7	1.081
7	8	1.077
8	9	1.073
9	10	1.068

5.2.2 Задания для самостоятельной работы

В данной работе требуется построить средствами языка Python и необходимых библиотек модели регрессий исследуемого ряда и сделать прогноз его значений на определенный период времени.

В ходе выполнения работы должны быть решены следующие задачи:

а) сформулировать словесный алгоритм программы;

б) построить блок-схему программы;

в) написать код программы;

г) сделать выводы и составить отчет.

Индивидуальные варианты задания и файлы с данными выдаются преподавателем.

5.3 Контрольные вопросы

1. Что такое аппроксимация?
2. Что такое регрессия?
3. Какие существуют виды регрессий?
4. Основные модели регрессий.
5. Как осуществляется прогнозирование уровней временного ряда?
6. Сформулировать метод наименьших квадратов и записать его математическую модель.
7. Приведение нелинейной модели регрессии к линейной.
8. В каких библиотеках Python реализована регрессия?
9. Как определяется точность регрессии?

Тема №6. Минимизация функции методом наискорейшего градиентного спуска

6.1 Краткие теоретические сведения

Рассмотрим задачу поиска минимума (максимума) функции нескольких переменных.

Пусть имеется некоторая функция $f(x)$, где $x = (x_1, x_2, ..., x_n)^T$ представляет собой вектор-столбец. Требуется найти такую точку, в которой функция достигается своего минимума (максимума) в некоторой области.

Одним из методов решения данной задачи является метод градиентного спуска [11], основанный на линейной аппроксимация функции в окрестности точки.

Запишем первые два члена разложения функции в ряд Тейлора в окрестности точки $x^{(k)}$:

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}).$$

В многомерном случае x представляет собой n-мерный вектор, тогда

$$f(x) \approx f(x^{(k)}) + \nabla f(x^{(k)})^T (x - x^{(k)}). \qquad (6.1)$$

Найдем точку $x^{(k+1)}$ в окрестности $x^{(k)}$, в которой выполняется условие

$$f(x^{(k+1)}) < f(x^{(k)}).$$

Это условие выполняется в том случае, если второе слагаемое в (6.1) меньше нуля. Данное слагаемое является скалярным произведением двух векторов. Скалярное произведение отрицательно, если вектора разнонаправлены и минимально, когда угол между векторами равен π.

Если представить вектор $x^{(k+1)} - x^{(k)}$ в виде $-\alpha \nabla f(x^{(k)})$, то для следующей точки спуска $x^{(k+1)}$ можно получить:

$$x^{(k+1)} = x^{(k)} - \alpha \nabla f(x^{(k)}). \qquad (6.2)$$

Вектор $-\nabla f(x)$ представляет собой антиградиент функции в данной точке. Вектор антиградиента направлен в сторону наискорейшего уменьшения функции и перпендикулярен касательной к линии уровня в данной точке.

Если далее разложить функцию в ряд Тейлора в окрестности найденной точки $x^{(k+1)}$, то мы сможем найти следующую точку $x^{(k+2)}$, в которой значение функции меньше значения в точке $x^{(k+1)}$. Повторяя эту процедуру, мы получим последовательность точек, в которых функция убывает.

Число α в (6.2) задает шаг от текущей точки к последующей в направлении антиградиента. Шаг α может быть постоянным и переменным. В методе наискорейшего спуска вычисляется оптимальная длина шага α_k, приводящая к минимуму функцию $f(x^{(k+1)})$. По необходимому условию экстремальности α_k является решением уравнения

$$\frac{d}{d\alpha} f(x^{(k)} - \alpha \nabla f(x^{(k)})) = 0. \qquad (6.3)$$

Т.о. получаем итерационную процедуру нахождения координат точек спуска, в которых функция на каждом шаге достигает локального минимума.

Условие остановки процедуры поиска точек спуска можно задать двумя способами. По величине градиента условие остановки имеет вид:

$$\left\| \text{grad} f(x^{(k+1)}) \right\| \le \varepsilon, \qquad (6.4)$$

а по расстоянию между соседними точками спуска:

$$\left\| x^{(k+1)} - x^{(k)} \right\| \le \varepsilon \ . \qquad (6.5)$$

Главным недостатком градиентных методов является низкая скорость сходимости в случае функций «овражного» типа.

6.2 Практическая работа

Цель работы: получить навыки применения языка Python для решения задач безусловной минимизации функции методом наискорейшего градиентного спуска.

6.2.1 Пример выполнения работы

Рассмотрим задачу нахождения минимума функции $f(x) = x^2 + 3y^2 - x - 2y$ методом наискорейшего градиентного спуска.

В этом методе используется градиент функции, представляющий собой частные производные по каждой координате. Для их аналитического вычисления будем применять библиотеку SymPy [12]. Данная библиотека также предназначена для решения уравнений, в том числе дифференциальных, интегрирования, нахождения пределов, разложения функции в ряд и других символьных вычислений.

Загружаем необходимые библиотеки:

```
import sympy as sp
import numpy as np
```

Чтобы проводить символьные вычисления с помощью библиотеки SymPy необходимо объявить сами символьные переменные. Для этого используется функция `symbols()`:

```
x, y, alpha = sp.symbols('x, y, alpha')
```

Далее объявляем функцию, минимум корой мы будем искать:

```
def func(x,y):
    return x**2+3*y**2-x-2*y
```

В данном примере минимизируется функция двух переменных, поэтому для нахождения градиента нам понадобятся две частные производные. Для их вычисления воспользуемся функцией `diff()` из библиотеки SymPy.

Объявляем две функции, вычисляющие производные по x и y

```
def dx(x,y):
    return sp.diff(func(x,y),x)

def dy(x,y):
    return sp.diff(func(x,y),y)
```

Далее задаем точность метода `Epsilon`, а также начальные координаты `x_Old` и `y_Old`. Создаем три пустых списка, в которые мы будем записывать необходимые нам значения: `list_x_new` – список значений координат x новой точки, `list_y_new` – список значений координат y новой точки, `list_alpha` – список значений шага α_k, `list_func` – список значений функции в найденных точках спуска:

```
Epsilon = 0.8
x_Old = 5
y_Old = 5
k = 0
list_x_new = []
list_y_new = []
list_alpha = []
list_func = []
```

Теперь переходим к самому алгоритму наискорейшего спуска. Этот алгоритм запишем в цикле `while` с условием (6.4). Для этого зададим функцию условия остановки:

```
def stop_condition(x_Old,y_Old):
    c = sp.sqrt(dx(x,y)**2 + dy(x,y)**2)
    return c.evalf(3, subs={x:x_Old, y:y_Old}) > Epsilon
```

Метод evalf(n) преобразует значение к виду десятичной дроби с n знаками после запятой, subs используется для подстановки числовых значений вместо символьных переменных.

Далее начинается блок команд цикла while. В нем в переменные a и b записываем новые координаты точки спуска.

```
while stop_condition(x_Old,y_Old):
    a = x_Old \
       - alpha*dx(x,y).evalf(3, subs={x:x_Old, y:y_Old})
    b = y_Old \
       - alpha*dy(x,y).evalf(3, subs={x:x_Old, y:y_Old})
```

После этого решается уравнение $f'(a,b)=0$, при этом производная берется по alpha. Корнем этого уравнения является оптимальный шаг, обозначенный alpha_opt. Решение уравнений вида $f(x)=0$ в библиотеке SymPy осуществляется методом solve():

```
    # Получаем список корней
    alpha_opt = sp.solve(
                   sp.diff(func(a, b), alpha), alpha)
    # Выбираем максимальный положительный корень
    alpha_opt = max([i.evalf(3) for i in
                    alpha_opt if i>0])
```

Затем с найденным оптимальным шагом вычисляются новые значения координат x_New и y_New. Значения x_New, y_New и alpha_opt добавляются в списки list_x_new, list_y_new и list_alpha:

```
    x_New = x_Old \
       - alpha_opt*dx(x,y).evalf(3,subs={x:x_Old,y:y_Old})
    y_New = y_Old \
       - alpha_opt*dy(x,y).evalf(3,subs={x:x_Old,y:y_Old})
    x_Old = x_New
    y_Old = y_New
    list_x_new.append(x_New)
    list_y_new.append(y_New)
```

```
    list_alpha.append(alpha_opt)
    list_func.append(func(x_New, y_New))
    k = k + 1
```

На этом блок команд в цикле `while` завершается.

Далее выводим полученные значения точки минимума, число шагов и координаты точек спуска:

```
print('Точка минимума:',
      'Xmin = ', x_New,'; Ymin =', y_New)
print('Число шагов = ',k)
print('Значения шагов = ', list_alpha)
print('Значения новых x = ', list_x_new)
print('Значения новых y = ', list_y_new)
```

В результате выполнения написанного кода мы получим координаты точки минимума, количество шагов и списки промежуточных результатов на каждой итерации цикла:

```
Точка минимума: Xmin =  0.796 ; Ymin = 0.301
Число шагов =  3
Значения шагов =  [0.178, 0.421, 0.178]
Значения новых x =  [3.40, 0.960, 0.796]
Значения новых y =  [0.0215, 0.809, 0.301]
```

Теперь построим трехмерный график минимизируемой функции и график линий уровня с точками спуска.

Начинаем с загрузки необходимых библиотек.

Библиотека `matplotlib.cm` используется для создания цветовых переходов, а модуль `Axes3D` из `mpl_toolkits.mplot3d` применяется для построения трехмерных графиков. Для управления отображением делений на осях применяется модуль `matplotlib.ticker`:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.cm as cm
import matplotlib.ticker as ticker
```

Для построения трехмерного графика необходимо добавить параметр `projection = '3d'`:

```
fig = plt.figure(figsize = (20, 7))
ax1 = fig.add_axes([0,0,0.7,1], projection = '3d')
ax2 = fig.add_axes([0.6,0.08,0.35,0.8])
```

В методах `add_axes()` для каждой области для построения графика задаем отдельно ее размеры.

Создаем массивы значений координат x и y с помощью функции `linspace(a,b,n)`, где `a` и `b` – границы интервала, `n` – количество точек

```
x_val = np.linspace(-5,5, 150)
y_val = np.linspace(-5,5, 150)
```

Для построения трехмерного графика создадим эквидистантную сетку из точек с координатами `x_val` и `y_val`. Для этого используется функция `meshgrid()`, применяя которую, получаем два массива координат `X_arg` и `Y_arg`:

```
X_arg, Y_arg = np.meshgrid(x_val, y_val)
```

После этого строим поверхность с помощью функции `plot_surface()`. Параметры `rstride` и `cstride` отвечают за шаг, с которым берутся значения из массивов `X_arg` и `Y_arg`, при этом, чем значение меньше, тем плавнее будет прорисовка (1 минимум). Цветовая схема графика задается параметром `cmap`:

```
ax1.plot_surface(X_arg, Y_arg, func(X_arg,Y_arg),
                 rstride = 1,
                 cstride = 1,
                 cmap='RdGy')
ax1.tick_params(axis = 'both', labelsize = 22)
```

Теперь добавим на трехмерный график линии уровня.

Начинаем с сортировки значений функции в точках спуска по возрастанию. Эти значения соответствуют значениям линий уровня для каждой точки спуска. Далее с помощью функции `contour()`

строим контурный график. Параметр zdir отвечает за ось координат, вдоль которой будет проецироваться трехмерный график на плоскость, на которой будут прорисованы линии уровня. Параметр offset задает положение плоскости проекции вдоль оси, заданной параметром zdir, levels – значения проецируемых линий уровня:

```
level = sorted(list_func)
ax1.contour(X_arg,Y_arg, func(X_arg,Y_arg),
            zdir = 'z',
            offset = -20,
            cmap = 'RdGy',
            levels = level)
```

Подписи осей трехмерного графика зададим в виде текста. С помощью zaxis.set_major_locator по оси Z задаем шаг делений основной сетки равный 50:

```
ax1.zaxis.set_major_locator(ticker.MultipleLocator(50))
ax1.text(4.5, 6, 210, "func(x,y)", size=22)
ax1.text(1.5, -9.5, 0, "x", size=22)
ax1.text(8.5, -3, 0, "y", size=22)
```

Теперь построим второй график, непосредственно показывающий линии уровня и шаги по точкам спуска. Строим те же лини уровня:

```
ax2.contour(X_arg, Y_arg, func(X_arg, Y_arg),
            cmap='RdGy', levels = level)
```

Добавляем к линиям уровня прямые, изображающие шаги от предыдущей точки спуска к следующей:

```
ax2.plot(list_x_new,list_y_new, color = 'green',
         linestyle = '--')
```

Отдельно строим шаг от начальной точки к первой точке спуска:

Метод evalf(n) преобразует значение к виду десятичной дроби с n знаками после запятой, subs используется для подстановки числовых значений вместо символьных переменных.

Далее начинается блок команд цикла while. В нем в переменные a и b записываем новые координаты точки спуска.

```
while stop_condition(x_Old,y_Old):
    a = x_Old \
       - alpha*dx(x,y).evalf(3, subs={x:x_Old, y:y_Old})
    b = y_Old \
       - alpha*dy(x,y).evalf(3, subs={x:x_Old, y:y_Old})
```

После этого решается уравнение $f'(a,b)=0$, при этом производная берется по alpha. Корнем этого уравнения является оптимальный шаг, обозначенный alpha_opt. Решение уравнений вида $f(x)=0$ в библиотеке SymPy осуществляется методом solve():

```
    # Получаем список корней
    alpha_opt = sp.solve(
                   sp.diff(func(a, b), alpha), alpha)
    # Выбираем максимальный положительный корень
    alpha_opt = max([i.evalf(3) for i in
                    alpha_opt if i>0])
```

Затем с найденным оптимальным шагом вычисляются новые значения координат x_New и y_New. Значения x_New, y_New и alpha_opt добавляются в списки list_x_new, list_y_new и list_alpha:

```
    x_New = x_Old \
       - alpha_opt*dx(x,y).evalf(3,subs={x:x_Old,y:y_Old})
    y_New = y_Old \
       - alpha_opt*dy(x,y).evalf(3,subs={x:x_Old,y:y_Old})
    x_Old = x_New
    y_Old = y_New
    list_x_new.append(x_New)
    list_y_new.append(y_New)
```

Работа включает два задания.

В первом задании требуется средствами языка Python и необходимых библиотек реализовать алгоритм метода наискорейшего градиентного спуска. Результатом работы алгоритма должен быть график минимизируемой функции и график линий уровня с изображением шагов спуска Кроме графиков необходимо вывести количество шагов и значений координат точек спуска.

В ходе выполнения первого задания требуется:

а) сформулировать словесный алгоритм программы;

б) построить блок-схему программы;

в) написать код программы;

г) сделать выводы и составить отчет.

Во втором задании необходимо провести исследование влияния двух видов условий остановки (формулы 6.4 и 6.5) на количество шагов алгоритма.

Индивидуальные варианты задания:

1. $f(x) = 5x_1^2 + 3(x_2 - 1)^2 - x_1x_2 - 2x_2$, $x^{(0)} = (13, 10)$, $\varepsilon = 0.5$.
2. $f(x) = 3x_1^2 + (x_2 - 5)^2 - x_1x_2 - x_1$, $x^{(0)} = (-7, 15)$, $\varepsilon = 0.5$.
3. $f(x) = 3x_1^2 + (2x_2 - 3)^2 - x_1(x_2 - 1) - x_1$, $x^{(0)} = (7, 10)$, $\varepsilon = 0.5$.
4. $f(x) = 3x_1^2 + (x_2 + 3)^2 - 3x_1 - x_2$, $x^{(0)} = (-5, 3)$, $\varepsilon = 0.5$.
5. $f(x) = x_1^2 + (x_2 + 5)^2 - x_1x_2 - 2x_2$, $x^{(0)} = (5, 5)$, $\varepsilon = 0.5$.
6. $f(x) = (2x_1 - 1)^2 + (x_2 + 5)^2 - x_1x_2 - 2x_2$, $x^{(0)} = (15, 3)$, $\varepsilon = 0.5$.
7. $f(x) = (2x_1 - 1)^2 + (x_2 - 5)^2 - x_1x_2 - 2x_2$, $x^{(0)} = (15, 13)$, $\varepsilon = 0.5$.
8. $f(x) = (2x_1 - 5)^2 + (x_2 - 4)^2 - 2x_1x_2 - x_2$, $x^{(0)} = (2, 15)$, $\varepsilon = 0.5$.
9. $f(x) = (2x_1 + 5)^2 + (-3x_2 + 2)^2 - 3x_1x_2 - 2x_2$, $x^{(0)} = (4, 3)$, $\varepsilon = 0.5$.
10. $f(x) = (-2x_1 + 5)^2 + (3x_2 + 2)^2 - x_1x_2 - 2x_2$, $x^{(0)} = (5, 3)$, $\varepsilon = 0.5$.
11. $f(x) = (-2x_1 - 5)^2 + (-3x_2 + 2)^2 + x_1x_2 - 2x_2$, $x^{(0)} = (1, 4)$, $\varepsilon = 0.5$.
12. $f(x) = (-2x_1 + 7)^2 + (-3x_2 + 2)^2 + x_1x_2 + 2x_2$, $x^{(0)} = (1, 4)$, $\varepsilon = 0.5$.
13. $f(x) = (-2x_1 + 5)^2 + (x_2 + 2)^2 + x_1x_2 - 5x_2$, $x^{(0)} = (6, 4)$, $\varepsilon = 0.5$.
14. $f(x) = (5x_1 + 2)^2 + (-2x_2 + 3)^2 + x_1x_2 - x_2$, $x^{(0)} = (7, 7)$, $\varepsilon = 0.5$.
15. $f(x) = (5x_1 + 3)^2 + (4x_2 - 5)^2 + 4x_1x_2 - x_2$, $x^{(0)} = (15, 15)$, $\varepsilon = 0.5$.

6.3 Контрольные вопросы

1. Что такое градиент функции?
2. Как вычисляется следующая точка спуска в методе наискорейшего градиентного спуска?
3. Записать компоненты вектора градиента функции 3-х переменных.
4. Как вычисляется величина шага в методе наискорейшего градиентного спуска?
5. Записать условия остановки в методе наискорейшего градиентного спуска.
6. В чем заключается главный недостаток градиентных методов?
7. В каких библиотеках Python реализовано вычисление производных функций?
8. Что такое линии уровня функции?
9. Сделать графическую иллюстрацию метода наискорейшего градиентного спуска.

Тема №7. Решение оптимизационных задач с ограничениями

7.1 Краткие теоретические сведения

Оптимизационные задачи с ограничениями принадлежат к классу задач линейного программирования. Линейное программирование изучает методы нахождения экстремума линейной функции нескольких переменных при дополнительных ограничениях в виде линейных равенств (или неравенств), накладываемых на переменные. К данному классу задач относятся транспортные задачи, распределительные задачи, задачи календарного планирования и т.п.

Математическая модель задачи линейного программирования включает в себя систему линейных ограничений (равенств или неравенств) отражающих условие задачи, а также линейную функцию выражающую цель задачи. Такая функция называется целевой функцией.

В общем виде задача линейного программирования может быть сформулирована следующим образом: найти минимум (максимум) линейной функции [11, 13]

$$z = c_0 + c_1 x_1 + c_2 x_2 + ... + c_n x_n, \tag{7.1}$$

при условии, что переменные $x_1, x_2, ..., x_n$ не отрицательные и удовлетворяют системе ограничений

$$\begin{cases} a_{11} x_1 + a_{12} x_2 + ... + a_{1n} x_n \geq b_1, \\ a_{21} x_1 + a_{22} x_2 + ... + a_{2n} x_n \geq b_2, \\ ... \\ a_{m1} x_1 + a_{m2} x_2 + ... + a_{mn} x_n \geq b_m. \end{cases} \tag{7.2}$$

Точка, являющаяся решением задачи линейного программирования, располагается на границе области допустимых

решений. Эта область представляет собой многогранник, образованный неравенствами системы ограничений (7.2).

Одним из наиболее популярных методов решения задач линейного программирования является симплекс-метод. Идея симплекс-метода заключается в переборе вершин многогранника области допустимых решений. Данный перебор производится при условии последовательного улучшения целевой функции (7.1). В результате находится вершина многогранника, в которой целевая функция достигает максимума (или минимума).

7.2 Практическая работа

Цель работы: получить навыки решения задач линейного программирования с помощью средств и методов языка Python и библиотеки SciPy.

7.2.1 Пример выполнения работы

Решим с применением возможностей языка Python транспортную задачу.

Пусть имеются поставщики товаров с трех складов расположенных в разных городах. Объемы продукции на этих складах равны: $a_1 = 74$, $a_2 = 40$, $a_3 = 36$. В других трех городах находятся потребители, потребности которых в этих товарах равны соответственно $b_1 = 20$, $b_2 = 45$, $b_3 = 30$. Стоимости доставки c_{ij} от *i*-го поставщика к *j*-му потребителю представлены матрицей

$$C = \begin{pmatrix} 7 & 3 & 6 \\ 4 & 8 & 2 \\ 1 & 5 & 9 \end{pmatrix}.$$

Требуется составить такой план перевозок товара от поставщиков к потребителям, чтобы стоимость его перевозки была минимальной.

Если x_{ij} – количество перевозимого товара от i-го поставщика к j-му потребителю, то целевая функция для этой задачи примет вид

$$F = c_{11}x_{11} + c_{12}x_{12} + c_{13}x_{13} + ... = \sum_{i,j=1}^{3} c_{ij}x_{ij} \to \min .$$

Обозначим коэффициенты $c_{11} = c_1$, $c_{12} = c_2$, …, $c_{33} = c_9$, а переменные $x_{11} = x_1$, $x_{12} = x_2$, …, $x_{33} = x_9$. Тогда для целевой функции получим

$$F = c_1x_1 + c_2x_2 + c_3x_3 + ... = \sum_{i=1}^{9} c_ix_i \to \min .$$

Запишем ограничения задачи. Т.к. от каждого поставщика возможно поставить товара не больше, чем у него имеется, то

$$\begin{cases} x_1 + x_2 + x_3 \le 74, \\ x_4 + x_5 + x_6 \le 40, \\ x_7 + x_8 + x_9 \le 36. \end{cases}$$

При этом необходимо удовлетворить потребности каждого потребителя, т.о.

$$\begin{cases} x_1 + x_4 + x_7 = 20, \\ x_2 + x_5 + x_8 = 45, \\ x_3 + x_6 + x_9 = 30. \end{cases}$$

Тогда система ограничений задачи примет следующий вид:

$$\begin{cases} x_1 + x_2 + x_3 \le 74, \\ x_4 + x_5 + x_6 \le 40, \\ x_7 + x_8 + x_9 \le 36, \\ x_1 + x_4 + x_7 = 20, \\ x_2 + x_5 + x_8 = 45, \\ x_3 + x_6 + x_9 = 30, \\ x_i \ge 0, i = 1, ..., 9. \end{cases}$$

а целевая функция:

$$F = 7x_1 + 3x_2 + 6x_3 + 4x_4 + 8x_5 + 2x_6 + x_7 + 5x_8 + 9x_9 \to \min .$$

Для решения задач линейного программирования в Python существует несколько библиотек: PuLP, CVXOPT, SciPy.

Мы будем использовать модуль optimize из библиотеки SciPy [14]. Эту библиотеку отличает большее быстродействие и удобство ввода данных.

Из модуля optimize библиотеки SciPy импортируем функцию `linprog()`:

```
from scipy.optimize import linprog
```

Функция `linprog()` предназначена для решения задач линейного программирования и имеет вид:

```
scipy.optimize.linprog(c, A_ub=None, b_ub=None, A_eq=None,
b_eq=None, method='highs', etc.)
```

Её основными атрибутами являются: `c` – список коэффициентов целевой функции, `A_ub` – матрица коэффициентов условий-неравенств, `b_ub` – значения из правой части условий-неравенств, `A_eq` – матрица коэффициентов условий-равенств, `b_eq` – значения из правой части условий-равенств, `method` – метод решения задачи оптимизации.

Сначала мы создаем эти списки пустыми, а затем заполняем путем ввода значений с клавиатуры.

Для списка коэффициентов целевой функции вводим

```
c = []
c.append([float(i) for i in input('Введите коэффициенты'
        + 'целевой функции через запятую:').split(',')])
```

Команда append() добавляет к пустому списку список значений коэффициентов, которые мы вводим через запятую.

Функция split(',') разбивает строку, введенную с клавиатуры, на части по разделителю, указанному в скобках, и создает список символов из полученных частей.

Далее, создаем и заполняем матрицу коэффициентов условий-неравенств. Т.к. эта матрица двумерная, заполняем ее путем добавления в цикле строк к первоначальному пустому списку:

```
A_ub = []
N_ner = int(input('Количество ограничений-неравенств = '))
for j in range(N_ner):
    coeff_a_ub = input('Введите коэффициенты ' + str(j+1)
          + '-го неравенства через запятую: ').split(',')
    A_ub.append([float(i) for i in coeff_a_ub])
```

Аналогично заполняем остальные списки и матрицы:

```
b_ub = []
coeff_b_ub = input('Коэффициенты правой части '
    + 'ограничений-неравенств через запятую: ').split(',')
b_ub.append([float(i) for i in coeff_b_ub])

A_eq = []
N_rav = int(input('Количество ограничений-равенств= '))
for j in range(N_rav):
    coeff_a_eq = input('Введите коэффициенты ' + str(j+1)
            + '-го равенства через запятую: ').split(',')
    A_eq.append([float(i) for i in coeff_a_eq])

b_eq=[]
```

```
coeff_b_eq = input('Коэффициенты правой части '
      + 'ограничений-равенств через запятую: ').split(',')
b_eq.append([float(i) for i in coeff_b_eq])

print(linprog(c, A_ub, b_ub, A_eq, b_eq))
```

Особенностью функции `linprog()` является то, что для каждого уравнения системы ограничений требуется вводить коэффициенты всех переменных, входящих в целевую функцию. Если в уравнении системы ограничений не входит какая-либо переменная, то для неё вводится коэффициент равный нулю. Т.е. для рассматриваемой задачи система ограничений, коэффициенты которой будут вводиться в матрицы `A_ub` и `A_eq`, примет вид:

$$\begin{cases} x_1 + x_2 + x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 \le 74, \\ 0x_1 + 0x_2 + 0x_3 + x_4 + x_5 + x_6 + 0x_7 + 0x_8 + 0x_9 \le 40, \\ 0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + x_7 + x_8 + x_9 \le 36, \\ x_1 + 0x_2 + 0x_3 + x_4 + 0x_5 + 0x_6 + x_7 + 0x_8 + 0x_9 = 20, \\ 0x_1 + x_2 + 0x_3 + 0x_4 + x_5 + 0x_6 + 0x_7 + x_8 + 0x_9 = 45, \\ 0x_1 + 0x_2 + x_3 + 0x_4 + 0x_5 + x_6 + 0x_7 + 0x_8 + x_9 = 30, \\ x_i \ge 0, i = 1, ..., 9. \end{cases}$$

Вводя коэффициенты этой системы ограничений, в результате работы кода программы получим:

```
con: array([8.03400049e-08, 1.98486838e-07,
            1.27598781e-07])
fun: 214.99999944270834
message: 'Optimization terminated successfully.'
nit: 5
slack: array([29.00000022, 10.00000009, 16.0000001 ])
status: 0
success: True
x: array([1.17814759e-08, 4.49999998e+01, 8.01637322e-09,
4.06707740e-08,       9.61431590e-09,       2.99999999e+01,
1.99999999e+01, 3.01544961e-08, 5.79416775e-09])
```

Итогом работы программы является набор данных, содержащий следующую основную информацию: `x` – вектор решения (значения переменных, при которых достигается цель задачи); `fun` – оптимальное значение целевой функции; `slack` – остатки в условиях-неравенствах (в рассматриваемой задаче это остатки товара на складах).

7.2.2 Задания для самостоятельной работы

В данной работе требуется решить задачу линейного программирования средствами языка Python.

В ходе выполнения работы требуется:

а) сформулировать словесный алгоритм программы;

б) построить блок-схему программы;

в) написать код программы;

г) сделать выводы и составить отчет.

Условие задачи для всех вариантов одинаковое: найти оптимальный план перевозок груза с трех складов в четыре пункта назначения. Конкретные значения запасов, потребностей и тарифов на перевозки представлены в индивидуальных вариантах.

Индивидуальные варианты задания:

1. Запасы на складах: 38 т, 50 т, 15 т. Потребности в пунктах назначения: 15 т, 25 т, 10 т, 8 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 10 & 7 & 9 & 3 \\ 5 & 11 & 8 & 6 \\ 6 & 10 & 12 & 3 \end{pmatrix}.$$

2. Запасы на складах: 48 т, 30 т, 15 т. Потребности в пунктах назначения: 10 т, 25 т, 10 т, 18 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 8 & 7 & 6 & 3 \\ 15 & 11 & 8 & 6 \\ 6 & 10 & 16 & 3 \end{pmatrix}.$$

3. Запасы на складах: 38 т, 20 т, 45 т. Потребности в пунктах назначения: 10 т, 25 т, 10 т, 18 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 8 & 7 & 7 & 3 \\ 5 & 11 & 18 & 6 \\ 6 & 10 & 12 & 3 \end{pmatrix}.$$

4. Запасы на складах: 28 т, 57 т, 15 т. Потребности в пунктах назначения: 25 т, 25 т, 10 т, 8 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 10 & 7 & 9 & 3 \\ 15 & 11 & 8 & 6 \\ 6 & 10 & 12 & 13 \end{pmatrix}.$$

5. Запасы на складах: 36 т, 50 т, 25 т. Потребности в пунктах назначения: 17 т, 20 т, 10 т, 18 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 10 & 7 & 9 & 13 \\ 5 & 11 & 8 & 6 \\ 16 & 10 & 12 & 3 \end{pmatrix}.$$

6. Запасы на складах: 38 т, 47 т, 10 т. Потребности в пунктах назначения: 15 т, 25 т, 10 т, 28 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 10 & 7 & 9 & 3 \\ 5 & 7 & 18 & 6 \\ 6 & 10 & 12 & 3 \end{pmatrix}.$$

7. Запасы на складах: 38 т, 50 т, 15 т. Потребности в пунктах назначения: 12 т, 15 т, 10 т, 28 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 6 & 7 & 9 & 3 \\ 5 & 11 & 8 & 11 \\ 8 & 10 & 12 & 3 \end{pmatrix}.$$

8. Запасы на складах: 48 т, 54 т, 10 т. Потребности в пунктах назначения: 25 т, 30 т, 10 т, 18 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 10 & 12 & 9 & 3 \\ 5 & 9 & 8 & 6 \\ 6 & 10 & 8 & 3 \end{pmatrix}.$$

9. Запасы на складах: 38 т, 50 т, 35 т. Потребности в пунктах назначения: 15 т, 25 т, 10 т, 11 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 10 & 7 & 9 & 13 \\ 15 & 11 & 8 & 6 \\ 6 & 8 & 12 & 3 \end{pmatrix}.$$

10. Запасы на складах: 58 т, 30 т, 15 т. Потребности в пунктах назначения: 5 т, 25 т, 10 т, 18 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 3 & 7 & 9 & 3 \\ 5 & 11 & 18 & 6 \\ 5 & 10 & 12 & 3 \end{pmatrix}.$$

11. Запасы на складах: 38 т, 50 т, 22 т. Потребности в пунктах назначения: 3 т, 15 т, 20 т, 8 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 10 & 7 & 5 & 3 \\ 3 & 11 & 8 & 6 \\ 6 & 10 & 8 & 8 \end{pmatrix}.$$

12. Запасы на складах: 18 т, 50 т, 35 т. Потребности в пунктах назначения: 9 т, 25 т, 9 т, 8 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 10 & 7 & 9 & 3 \\ 15 & 11 & 8 & 16 \\ 6 & 10 & 12 & 3 \end{pmatrix}.$$

13. Запасы на складах: 38 т, 44 т, 18 т. Потребности в пунктах назначения: 16 т, 25 т, 20 т, 8 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 10 & 4 & 9 & 3 \\ 5 & 11 & 8 & 6 \\ 6 & 9 & 9 & 3 \end{pmatrix}.$$

14. Запасы на складах: 38 т, 50 т, 35 т. Потребности в пунктах назначения: 15 т, 12 т, 10 т, 28 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 4 & 7 & 9 & 3 \\ 5 & 6 & 8 & 8 \\ 6 & 10 & 12 & 3 \end{pmatrix}.$$

15. Запасы на складах: 28 т, 48 т, 15 т. Потребности в пунктах назначения: 15 т, 15 т, 10 т, 8 т. Тарифы на перевозки (тыс. руб.):

$$C = \begin{pmatrix} 7 & 17 & 9 & 3 \\ 5 & 11 & 8 & 6 \\ 6 & 8 & 12 & 3 \end{pmatrix}.$$

7.3 Контрольные вопросы

1. Записать математическую формулировку задачи линейного программирования в общем виде.
2. Что такое область допустимых решений?
3. Как формируется область допустимых решений?
4. Как решается задача линейного программирования симплекс-методом?
5. Что такое целевая функция?
6. Геометрический смысл симплекс-метода.
7. Какое решение называется допустимым?
8. Записать математическую формулировку транспортной задачи.
9. Какой многогранник называется выпуклым?

Тема №8. Решение оптимизационных задач на графах

8.1 Краткие теоретические сведения

Для представления и анализа различных структур, имеющих связи между входящими в них элементами применяется особая математическая конструкция, называемая графом [15].

Граф представляет собой множество точек (узлов или вершин), часть из которых или все соединены линиями, изображающими связи между этими точками. Эти линии называются ребрами.

Существует два основных вида графов: ориентированные и неориентированные. В ориентированном графе линии, соединяющие вершины, имеют направление от одной из них к другой. В неориентированном графе линии не имеют направления.

Граф называется простым, если каждую пару вершин соединяет не более чем одно ребро. Граф называется мультиграфом, если хотя бы одну пару вершин соединяет более чем одно ребро.

Граф может быть задан двумя дискретными множествами – множеством вершин, и множеством ребер.

Если две вершины соединены ребром, они называются смежными. Граф можно задать, если для всех пар вершин указать наличие или отсутствие между ними ребра. Это можно реализовать с помощью матрицы, элемент которой c_{ij} принимает значение 1, если вершина i смежна вершине j, и значение равное 0, если вершины не соединены. Такая матрица называется матрицей смежности. Если в графе n вершин, то матрица смежности имеет размер $n \times n$.

Другой способ задания графа реализуется с помощью матрицы инцидентности. Ребро инцидентно вершине, если соединяет ее с какой-либо другой. Матрица инцидентности состоит из n строк (n – число вершин) и m столбцов (m – число ребер). Элемент данной матрицы a_{ij} принимает значение 1, если ребро j инцидентно вершине i, и значение равное 0, если не инцидентно.

Для анализа графов и извлечения из них необходимой информации разработаны различные методы и алгоритмы.

Одним из типов задач, решаемых с помощью графов, являются задачи поиска кратчайших путей между вершинами, построение сетей минимальной длины и т.п. Решение подобных задач основано на алгоритмах перебора вершин и ребер.

Для решения оптимизационных задач на графах с помощью языка Python мы будем использовать библиотеку NetworkX [16]. В данной библиотеке возможно создание различных типов графов с помощью функций:

а) `Graph()` – простой неориентированный граф, при этом узлы могут быть соединённые сами с собой;

б) `DiGraph()` – ориентированный граф;

в) `MultiGraph()` – мультиграф, в нем возможно существование пар вершин, которые соединены более чем одним ребром, либо более чем двумя дугами противоположных направлений;

г) `MultiDiGraph()` – ориентированный мультиграф.

Составляющими элементами любого графа являются вершины (`node`) и ребра (`edge`), которые имеют свои собственные идентификаторы, а также различные атрибуты.

В библиотеке NetworkX граф может быть создан тремя методами, отличающимися способами задания вершин и дуг: с использованием генератора графов с предустановленной топологией (полный граф, циклические графы и т.д.); загрузкой данных из структуры (матрица смежности, матрица инцидентности и др.); последовательным добавлением вершин (узлов) и (или) ребер.

Функция `add_node()` применяется для добавления одной вершины, а функция `add_nodes_from()`, аргументом которой является список, кортеж, строки из файла или узлы другого графа, применяется для добавления сразу нескольких вершин. Чтобы удалить вершину используется функция `remove_node()`.

Для добавления ребер между парой вершин применяется функция `add_edge()`, а для удаления ребра функция `remove_edge()`. Параметр `weight` функции `add_edge()` задает «вес» ребра. Данный параметр не является фиксированным атрибутом функции `add_edge()`, его значение и название задается пользователем. В алгоритмах библиотеки NetworkX, например, в алгоритмах для вычисления положения узлов при визуализации графа, используется фиксированный атрибут `weight`, которому по умолчанию присвоено имя `weight='weight'`. Поэтому, если для

названия «весов» ребер использовать какое-либо другое имя, то в алгоритмах, в которых присутствует атрибут `weight`, необходимо обязательно указывать выбранное имя `weight='имя'`.

Функция `degree('A', weight='weight')` возвращает сумму весов ребер непосредственно соединенных с вершиной «A». Если параметр `weight` не указан, то веса ребер считаются равными 1, и результатом будет число ребер, непосредственно соединенных с вершиной «A».

Чтобы визуализировать полученный граф, в библиотеке NetworkX существуют разные способы, например:

а) `draw()` – самый простой способ, без детальной настройки;

б) `draw_networkx()` – визуализация с дополнительными параметрами;

в) `draw_networkx_nodes()` – строит все узлы;

г) `draw_networkx_edges()` – строит все ребра;

д) `draw_circular()` – располагает узлы по кругу.

В примере 8.1 показано построение графа с вершинами «A», «B», «C», «D» и простой визуализацией с помощью функции `draw()`.

Пример 8.1. Создание и визуализация графа.

```
In[1]:
import networkx as nx

G = nx.Graph()
G.add_node('A')
G.add_node('B')
G.add_node('C')
G.add_node('D')
G.add_edge('A', 'B', weight = 2)
G.add_edge('A', 'D', weight = 3)
G.add_edge('C', 'D', weight = 2)
G.add_edge('B', 'C', weight = 2)
G.add_edge('B', 'D', weight = 1)

nx.draw(G,  node_size = [900, 900, 900, 900],
        font_size=22,
```

```
        with_labels=True,
        node_color="y")
ax = plt.gca()
# Задаем поля области построения, чтобы не
# обрезались узлы:
ax.margins(0.1)
```

Результат работы кода представлен на рисунке 8.1.

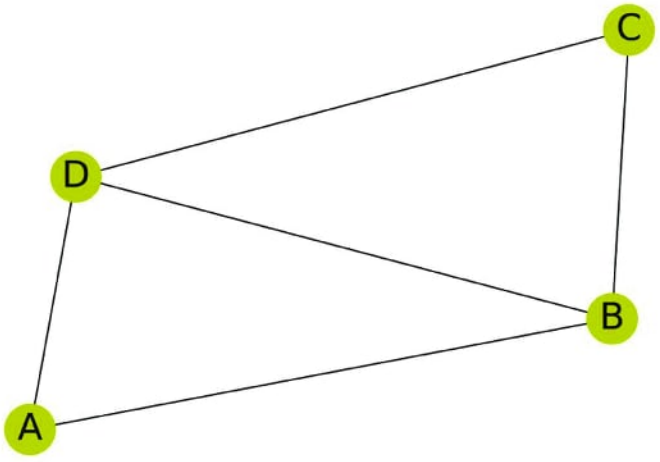


Рисунок 8.1 – Визуализация графа из примера 8.1

Этап добавления вершин можно пропустить и сразу перейти к добавлению ребер. Функция `add_edge('A','B')` автоматически добавит какую-либо из вершин «A» и (или) «B», если этой вершины еще нет.

По умолчанию функция `draw()` для расчета расположения узлов графа применяет алгоритм `spring_layout`. При этом при повторных запусках кода расположение вершин будет меняться. Чтобы зафиксировать их положение используется атрибут `pos` (представляет собой словарь), в который значения координат вершин передаются или вручную, или с помощью функции `get_node_attributes()`.

В примере 8.2 показано построение графа с заданными координатами вершин и его визуализацией с помощью метода `draw_networkx()`.

Пример 8.2. Построение взвешенного графа с фиксированным положением узлов.

```
In[2]:
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()

G.add_node('A', coordinate = (0,0))
G.add_node('B', coordinate = (0.4,0.6))
G.add_node('C', coordinate = (1,0.5))
G.add_node('D', coordinate = (0.8,0.1))
G.add_edge('A', 'B', weight = 1)
G.add_edge('A', 'C', weight = 2)
G.add_edge('A', 'D', weight = 4)

pos = nx.get_node_attributes(G,'coordinate')

labels = nx.get_edge_attributes(G,'weight')

nx.draw_networkx(G,
                 pos,
                 node_size = [800, 800, 800, 800],
                 font_size = 20,
                 with_labels = True,
                 node_color = "y")

nx.draw_networkx_edge_labels(G, pos,
                             font_size=16,
                             edge_labels=labels)

ax = plt.gca()
ax.margins(0.1)
plt.axis("off")
```

Результат выполнения кода представлен на рисунке 8.2.

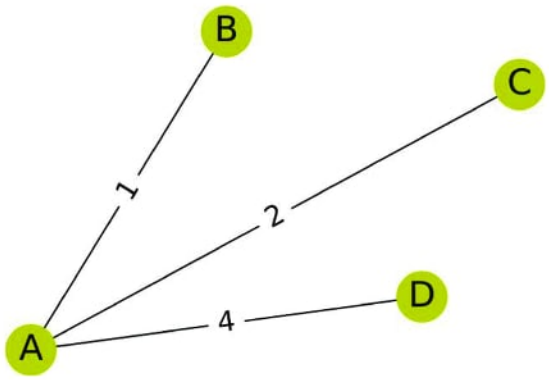


Рисунок 8.2 – Визуализация взвешенного графа с фиксированным положением вершин

8.2 Практическая работа

Цель работы: получить навыки решения оптимизационных задач на графах с помощью средств и методов языка Python и библиотеки NetworkX.

8.2.1 Пример выполнения работы

Решим следующую задачу.

Имеется 7 населенных пунктов, соединенных дорогами с известными расстояниями между соединенными городами. Требуется выполнить следующие задания:

а) найти кратчайший путь и его длину между городом №1 и городом №7;

б) проложить вдоль дорог волоконно-оптическую линию связи, соединяющую все города и имеющую минимальную длину.

Начинаем выполнение задания с импорта библиотеки для работы с графами NetworkX и создаем пустой граф `G`:

```
import networkx as nx
import matplotlib.pyplot as plt

G=nx.Graph()
```

В этот граф добавим вершины с их координатами из заранее созданного списка Uzly:

```
Uzly=[('1', {'koord':(0,0)}),
      ('2', {'koord':(0,1)}),
      ('3', {'koord':(0.5,0)}),
      ('4', {'koord':(0.5,1)}),
      ('5', {'koord':(1,0)}),
      ('6', {'koord':(1,1)}),
      ('7', {'koord':(1.5,0.5)})
]
```

Заранее создадим список (Dlina) ребер с их весами. Веса назовем length:

```
Dlina=[('1', '2', {'length':15}),
       ('1', '3', {'length':12}),
       ('2', '3', {'length':3}),
       ('2', '4', {'length':6}),
       ('2', '7', {'length':30}),
       ('3', '5', {'length':4}),
       ('4', '5', {'length':8}),
       ('4', '7', {'length':5}),
       ('5', '6', {'length':2}),
       ('6', '7', {'length':5})
]
```

Далее с помощью функций add_nodes_from() и add_edges_from() добавим из созданных списков в граф узлы и ребра:

```
G.add_nodes_from(Uzly)
G.add_edges_from(Dlina)
```

Чтобы положения вершин были зафиксированы, используем их координаты при визуализации графа функцией draw_networkx(). Для этого в данной функции необходимо указать

атрибут pos, содержащий координаты вершин. В данный атрибут координаты вершин передаем функцией get_node_attributes():

```
pos = nx.get_node_attributes(G, 'koord')
```

В атрибут labels запишем значения весов ребер:

```
labels = nx.get_edge_attributes(G,'length')
```

После этого визуализируем граф и для каждого ребра добавим подпись, показывающую его вес:

```
nx.draw_networkx(
          G,
          pos,
          node_size = [800, 800, 800, 800, 800, 800, 800],
          font_size=20,
          with_labels=True,
          node_color='y'
)

nx.draw_networkx_edge_labels(G,
                             pos,
                             font_size=16,
                             edge_labels=labels)
```

Далее для поиска кратчайшего пути между вершинами №1 и №7 применяем функцию shortest_path(), а для вычисления его длины – функцию shortest_path_length():

```
print('кратчайший путь между вершинами №1 и №7 = ',
      nx.shortest_path(G, '1', '7', weight='length'))

print('длина кратчайшего пути между вершинами №1 и №7 = ',
      nx.shortest_path_length(G,'1', '7',
      weight='length'))
```

```
ax = plt.gca()
ax.margins(0.1)
plt.axis("off")
```

В результате получим:

```
кратчайший путь между вершинами №1 и №7 = ['1', '3', '5',
'6', '7']
длина кратчайшего пути между вершинами №1 и №7 = 23
```

Получившийся граф представлен на рисунке 8.3.

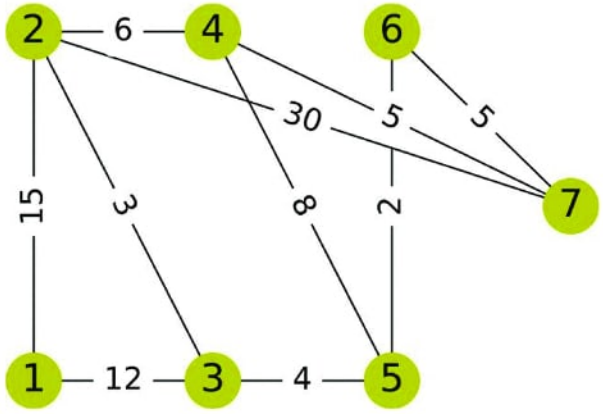


Рисунок 8.3 – Расположение городов и сеть дорог между ними

Построим теперь сеть минимальной длины. Для этого используем функцию `minimum_spanning_tree()`, в ней в качестве атрибута `weight` мы используем принятое нами обозначение весов ребер `length`:

```
G_min = nx.minimum_spanning_tree(G, weight = 'length')

nx.draw_networkx(
          G_min,
          pos,
          node_size = [800, 800, 800, 800, 800, 800, 800],
          font_size = 20,
          node_color = 'y'
)
```

```
labels_min = nx.get_edge_attributes(G_min, 'length')

nx.draw_networkx_edge_labels(G_min,
                             pos,
                             font_size = 16,
                             edge_labels = labels_min)

ax = plt.gca()
ax.margins(0.1)
plt.axis("off")
```

В результате получим сеть минимальной длины, которая представлена на рисунке 8.4.

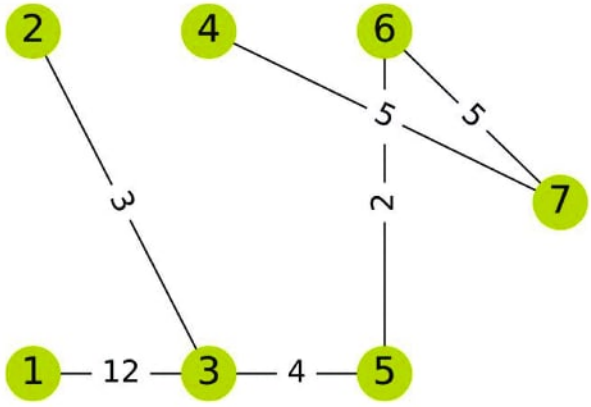


Рисунок 8.4 – Сеть минимальной длины

8.2.2 Задания для самостоятельной работы

В данной работе с помощью алгоритмов оптимизации на графах, представленных в библиотеке NetworkX, решается задача нахождения минимального пути и сети минимальной протяженности.

В ходе выполнения работы должны быть решены следующие задачи:

а) сформулировать словесный алгоритм программы;

б) построить блок-схему программы;

в) написать код программы;

г) сделать выводы и составить отчет.

Условие задачи для всех вариантов одинаковое: имеется 8 населенных пунктов: *A*, *B*, *C*, *D*, *E*, *F*, *G*, *H*. Населенные пункты соединены дорогами с известными расстояниями между соединенными городами. Требуется:

а) найти кратчайший путь и его длину между городами *E* и *F*;

б) проложить вдоль дорог волоконно-оптическую линию связи, соединяющую все города и имеющую минимальную длину.

Конкретные значения координат городов и длин дорог, соединяющих города, представлены в индивидуальных вариантах.

Индивидуальные варианты задания:

1. Координаты городов: $A=(0,0)$, $B=(3,0)$, $C=(0,6)$, $D=(5,5)$, $E=(3,9)$, $F=(6,1)$, $G=(2,5)$, $H=(8,8)$. Длины дорог: $S_{AB}=5$, $S_{BC}=10$, $S_{AC}=7$, $S_{DB}=8$, $S_{EG}=8$, $S_{EC}=9$, $S_{FH}=3$, $S_{CH}=9$, $S_{DF}=4$, $S_{DG}=3$, $S_{GH}=2$, $S_{GB}=2$.

2. Координаты городов: $A=(0,3)$, $B=(5,5)$, $C=(2,0)$, $D=(3,5)$, $E=(1,0)$, $F=(5,3)$, $G=(1,5)$, $H=(4,1)$. Длины дорог: $S_{AB}=5$, $S_{BC}=7$, $S_{AC}=12$, $S_{DF}=5$, $S_{EG}=8$, $S_{FG}=11$, $S_{FH}=7$, $S_{CH}=9$, $S_{AG}=14$, $S_{DG}=3$, $S_{GH}=2$, $S_{AC}=2$.

3. Координаты городов: $A=(1,0)$, $B=(0,4)$, $C=(1,7)$, $D=(3,0)$, $E=(3,4)$, $F=(6,1)$, $G=(6,5)$, $H=(3,6)$. Длины дорог: $S_{AB}=2$, $S_{BC}=4$, $S_{AC}=10$, $S_{DF}=5$, $S_{EG}=8$, $S_{FH}=7$, $S_{CH}=5$, $S_{AG}=14$, $S_{AE}=14$, $S_{BD}=3$, $S_{GH}=2$, $S_{AC}=2$.

4. Координаты городов: $A=(1,2)$, $B=(0,4)$, $C=(5,4)$, $D=(5,0)$, $E=(3,0)$, $F=(6,6)$, $G=(6,2)$, $H=(3,6)$. Длины дорог: $S_{AB}=2$, $S_{BC}=11$, $S_{AC}=8$, $S_{BF}=7$, $S_{CE}=9$, $S_{EG}=8$, $S_{FH}=5$, $S_{CH}=5$, $S_{AG}=14$, $S_{BD}=3$, $S_{DE}=2$, $S_{AC}=2$.

5. Координаты городов: $A=(4,5)$, $B=(0,4)$, $C=(5,1)$, $D=(3,0)$, $E=(0,2)$, $F=(6,6)$, $G=(6,2)$, $H=(2,6)$. Длины дорог: $S_{AB}=7$, $S_{BC}=11$, $S_{AF}=5$, $S_{DF}=12$, $S_{CE}=9$, $S_{FH}=6$, $S_{CH}=11$, $S_{FG}=4$, $S_{GF}=4$, $S_{BD}=3$, $S_{DE}=2$, $S_{AC}=2$.

6. Координаты городов: $A=(0,4)$, $B=(5,4)$, $C=(3,1)$, $D=(0,2)$, $E=(1,0)$, $F=(6,6)$, $G=(6,2)$, $H=(2,6)$. Длины дорог: $S_{BF}=3$,

$S_{BC}=11$, $S_{AF}=7$, $S_{AD}=12$, $S_{CE}=3$, $S_{FH}=6$, $S_{DH}=4$, $S_{CG}=4$, $S_{GF}=4$, $S_{BD}=3$, $S_{DE}=2$, $S_{AC}=2$.

7. Координаты городов: $A=(4,3)$, $B=(2,5)$, $C=(3,1)$, $D=(1,2)$, $E=(1,0)$, $F=(5,6)$, $G=(6,2)$, $H=(0,6)$. Длины дорог: $S_{AB}=12$, $S_{BC}=11$, $S_{AF}=7$, $S_{CD}=12$, $S_{CE}=3$, $S_{FH}=6$, $S_{DH}=4$, $S_{CG}=4$, $S_{GF}=4$, $S_{BD}=3$, $S_{DE}=2$, $S_{AC}=2$.

8. Координаты городов: $A=(0,0)$, $B=(2,1)$, $C=(0,6)$, $D=(6,4)$, $E=(3,9)$, $F=(7,0)$, $G=(2,5)$, $H=(8,7)$. Длины дорог: $S_{AB}=2$, $S_{BC}=6$, $S_{AC}=7$, $S_{BD}=5$, $S_{EG}=4$, $S_{EC}=9$, $S_{FH}=5$, $S_{CH}=15$, $S_{DF}=4$, $S_{DG}=3$, $S_{GH}=10$, $S_{GB}=2$.

9. Координаты городов: $A=(0,0)$, $B=(2,0)$, $C=(0,4)$, $D=(7,4)$, $E=(4,3)$, $F=(7,0)$, $G=(2,8)$, $H=(8,6)$. Длины дорог: $S_{AB}=2$, $S_{BC}=3$, $S_{AC}=3$, $S_{BD}=8$, $S_{EG}=4$, $S_{EC}=4$, $S_{FH}=5$, $S_{CH}=15$, $S_{DF}=4$, $S_{DG}=11$, $S_{GH}=10$, $S_{GB}=7$.

10. Координаты городов: $A=(1,0)$, $B=(0,3)$, $C=(3,0)$, $D=(7,4)$, $E=(6,0)$, $F=(8,2)$, $G=(2,8)$, $H=(8,8)$. Длины дорог: $S_{AB}=2$, $S_{BC}=3$, $S_{AC}=3$, $S_{BD}=8$, $S_{EG}=10$, $S_{EC}=4$, $S_{FH}=5$, $S_{CH}=15$, $S_{DF}=2$, $S_{DG}=11$, $S_{GH}=10$, $S_{GB}=7$.

11. Координаты городов: $A=(1,6)$, $B=(0,3)$, $C=(2,0)$, $D=(8,6)$, $E=(6,0)$, $F=(4,8)$, $G=(4,6)$, $H=(8,3)$. Длины дорог: $S_{AB}=2$, $S_{BC}=3$, $S_{AC}=5$, $S_{BD}=8$, $S_{EG}=7$, $S_{EC}=4$, $S_{FH}=5$, $S_{CH}=15$, $S_{DF}=2$, $S_{AF}=11$, $S_{GH}=6$, $S_{GB}=7$.

12. Координаты городов: $A=(1,6)$, $B=(1,3)$, $C=(5,0)$, $D=(8,6)$, $E=(7,0)$, $F=(5,8)$, $G=(4,6)$, $H=(8,3)$. Длины дорог: $S_{AB}=2$, $S_{BC}=3$, $S_{AC}=10$, $S_{BD}=12$, $S_{EG}=7$, $S_{EC}=4$, $S_{FH}=5$, $S_{CH}=5$, $S_{DF}=2$, $S_{AF}=7$, $S_{GH}=6$, $S_{GB}=7$.

13. Координаты городов: $A=(3,6)$, $B=(2,2)$, $C=(3,0)$, $D=(7,1)$, $E=(6,0)$, $F=(6,8)$, $G=(5,6)$, $H=(8,4)$. Длины дорог: $S_{AB}=2$, $S_{BC}=3$, $S_{GD}=7$, $S_{BD}=12$, $S_{EG}=7$, $S_{EC}=4$, $S_{FH}=5$, $S_{CH}=6$, $S_{DF}=2$, $S_{AF}=7$, $S_{AG}=6$, $S_{GB}=3$.

14. Координаты городов: $A=(0,6)$, $B=(1,3)$, $C=(0,0)$, $D=(8,0)$, $E=(6,0)$, $F=(6,8)$, $G=(4,6)$, $H=(7,4)$. Длины дорог: $S_{AB}=2$, $S_{BC}=3$, $S_{GD}=6$, $S_{BD}=14$, $S_{EG}=6$, $S_{EC}=3$, $S_{FH}=5$, $S_{CH}=5$, $S_{DF}=3$, $S_{AF}=8$, $S_{AG}=5$, $S_{GB}=4$.

15. Координаты городов: $A=(0,6)$, $B=(4,6)$, $C=(0,0)$, $D=(7,2)$, $E=(6,0)$, $F=(6,8)$, $G=(0,3)$, $H=(8,4)$. Длины дорог: $S_{AB}=2$, $S_{BC}=8$, $S_{GD}=7$, $S_{BD}=4$, $S_{EG}=7$, $S_{EC}=6$, $S_{FH}=5$, $S_{CH}=11$, $S_{DF}=2$, $S_{AF}=7$, $S_{AG}=2$, $S_{GB}=3$.

8.3 Контрольные вопросы

1. Что такое граф?
2. Что такое матрица смежности?
3. Что такое матрица инцидентности?
4. Какой граф называется простым?
5. Какой граф называется пустым?
6. Какой граф называется полным?
7. Какой граф называется неориентированным?
8. Записать алгоритм построения минимального остова.
9. Записать алгоритм нахождения минимального пути между двумя вершинами неориентированного графа.
10. Какой граф называется ориентированным?
11. Какой граф называется мультиграфом?
12. Что такое минимальный остов?

Заключение

Опыт преподавания различных дисциплин авторами данного пособия однозначно говорит о том, что для лучшего и более глубокого понимания обучающимися учебного материала крайне полезным является наличие лабораторного практикума наряду с лекционными и практическими занятиями.

В данном учебно-методическом пособии, представляющим собой компьютерный практикум для проведения лабораторных занятий, разобраны основы компьютерного моделирования на языке программирования Python и методы его применения для решения базовых оптимизационных задач.

Выбор в качестве инструмента решения представленных в пособии задач языка программирования Python обусловлен свободным доступом к нему и его многочисленным библиотекам, легкостью освоения, наличием большого и постоянно развивающегося сообщества программистов, а также его широким применением в области анализа данных и компьютерного моделирования.

Пособие адресовано студентам различных специальностей, применяющих методы математического и компьютерного моделирования, в особенности тем, для кого программирование не является профильной специальностью. Поэтому в пособии были приведены только необходимые основы алгоритмизации и не рассмотрены многие вопросы, касающиеся работы компьютерных программ, представления и хранения информации на компьютере, машинного языка и т.п. Однако успешное выполнение представленных в пособии заданий позволит сформировать у обучающихся базовые навыки программирования и компьютерного моделирования, вследствие чего данное пособие можно рекомендовать и тем, кто начинает изучение программирования на языке Python.

Библиографический список

1. Документация библиотеки NumPy. URL: https://numpy.org (дата обращения: 01.02.2023).
2. Документация библиотеки Matplotlib. URL: https:// matplotlib.org (дата обращения: 01.02.2023).
3. Шабанов П.А. Научная графика в python. Дата обновления: 31.08.2015. URL: https://github.com/whitehorn/Scientific_graphics_in _python (дата обращения: 01.02.2023).
4. Иванова Т.В. Численные методы в оптике. Учебное пособие. – СПб: Университет ИТМО, 2017 – 84 с.
5. Построение графиков в Python при помощи Matplotlib. URL: https://python-scripts.com (дата обращения: 20.02.2023).
6. Гмурман В.Е. Теория вероятностей и математическая статистика: учебное пособие для бакалавров / В.Е. Гмурман. – 12-е изд. – М.: Юрайт, 2014. – 478 с.
7. Анализ данных с помощью Pandas. URL: https:// pythonworld.ru/obrabotka-dannyx (дата обращения: 20.02.2023).
8. Документация библиотеки Pandas. URL: https://pandas. pydata.org (дата обращения: 01.03.2023).
9. Кильдишев Г.С., Френкель А.А. Анализ временных рядов и прогнозирование. – М: URSS. 2021, – 104 с.
10. Документация библиотеки Scikit-learn. URL: https://scikit-learn.org/stable (дата обращения: 05.03.2023).
11. Кочегурова Е.А. Теория и методы оптимизации / Е.А. Кочегурова; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2012. –157 с.
12. Документация библиотеки SymPy. URL: https://www. sympy.org/en/index.html (дата обращения: 10.03.2023).
13. Вентцель Е.С. Исследование операций: задачи, принципы, методология. – М.: Кнорус, 2014. – 192 с.
14. Документация библиотеки SciPy. URL: https://scipy.org (дата обращения: 12.03.2023).
15. Харари Ф. Теория графов / Ф. Харари – М. URSS, 2018. – 304 с.
16. Документация библиотеки NetworkX. URL: https:// networkx.org (дата обращения: 15.03.2023).