МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания по выполнению лабораторных работ

по дисциплине «Инструментальные средства визуализации данных»

для студентов направления подготовки
09.03.02 «Информационные системы и технологии»
Профиль «Прикладное программирование в интеллектуальных информационных системах»

Ставрополь, 2026

ВВЕДЕНИЕ

Цель освоения дисциплины: Методические указания к выполнению лабораторных работ составлены в соответствии с рабочей программой дисциплины «Инструментальные средства визуализации данных» и предназначены для студентов направления подготовки 09.03.02 «Информационные системы и технологии».

Целью освоения дисциплины «Инструментальные средства визуализации данных» является формирование профессиональных компетенций (ПК-2, ПК-6) будущего бакалавра по направлению подготовки 09.03.02 «Информационные системы и технологии». Для подготовки востребованных и конкурентоспособных на рынке труда бакалавров в области информационных систем и искусственного интеллекта, способных применять методы и средства визуализации данных для анализа данных, создания графиков и диаграмм, исследования статистических закономерностей и визуализации результатов исследований.

В рамках изучения дисциплины «Инструментальные средства визуализации данных» используются современные инструментальные средства, что позволяет подготовить компетентных специалистов, в соответствии с современными тенденциями и требованиями на рынке труда.

ЛАБОРАТОРНАЯ РАБОТА №1. ВИЗУАЛИЗАЦИЯ ДАННЫХ. УСТАНОВКА И ОПИСАНИЕ MATPLOTLIB

Цель: познакомиться с библиотекой Matplotlib, изучить варианты установки Matplotlib

Формируемые компетенции: ПК-2, ПК-6

Теоретическая часть

Matplotlib распространяется на условиях BSD-подобной лицензии. Библиотека поддерживает двумерную (2D) и трехмерную (3D) графику, а также анимированные рисунки.

Создаваемые изображения могут быть использованы в мультимедийных приложениях, научных проектах, а также различных документах, публикациях и веб-приложениях. Исторически библиотека формировалась под влиянием математического пакета Matlab, но являлась и является независимым от него проектом. Построенная на принципах ООП, библиотека также имеет процедурный интерфейс pylab, который предоставляет аналоги команд Matlab.

Пакет поддерживает многие виды диаграмм:

1. графики;
2. диаграммы разброса;
3. столбчатые диаграммы и гистограммы;
4. круговые диаграммы;
5. ствол-лист диаграммы;
6. контурные графики;
7. поля градиентов;
8. спектральные диаграммы и др.

При построении возможно указать оси координат, сетку, добавить аннотации, использовать логарифмическую шкалу или полярные координаты. Созданные изображения могут быть легко сохранены, в частности, в популярные форматы (JPEG, PNG и др.).

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими программными средствами разработки (выбрать один или несколько програмных продуктов для практической реализации задач лабораторной работы): интерпретатор Python (Anaconda).

Методика и порядок выполнения работы

Существует два основных варианта установки Matplotlib на ваш компьютер: в первом случае вы устанавливаете пакет Anaconda, в состав которого входит интересующая нас библиотека, во втором - устанавливаете Matplotlib самостоятельно, используя менеджер пакетов. Про установку Anaconda вы можете прочитать в статье “Python. Урок 1. Установка” (https://devpractice.ru/python-lesson-1-install/).

Для установки Matplotlib через менеджер пакетов pip введите в командной строке вашей операционной системы следующие команды:

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

Первая из них обновит ваш pip, вторая установит Matplotlib со всеми необходимыми зависимостями.

Для проверки того, что все установилось правильно, запустите интерпретатор Python и введите в нем следующее:

```
>>> import matplotlib
```

Если сообщений об ошибке не было, то значит библиотека Matplotlib установлена и ее можно использовать. После этого можете проверить версию библиотеки (она скорее всего будет отличаться от приведенной ниже):

```
>>> matplotlib.__version__
 '3.0.3'
```

Перед тем как углубиться в детали библиотеки Matplotlib, рассмотрим несколько примеров, изучив которые вы сможете использовать библиотеку для решения своих задач и получите интуитивное понимание принципов работы с этим инструментом. Если вы работаете в Jupyter Notebook, то для того, чтобы получать графики рядом с ячейками с кодом необходимо выполнить специальную magic команду после импорта Matplotlib:

import matplotlib.pyplot as plt

%matplotlib inline

Результат работы будет выглядеть так, как показано на рисунке ниже.

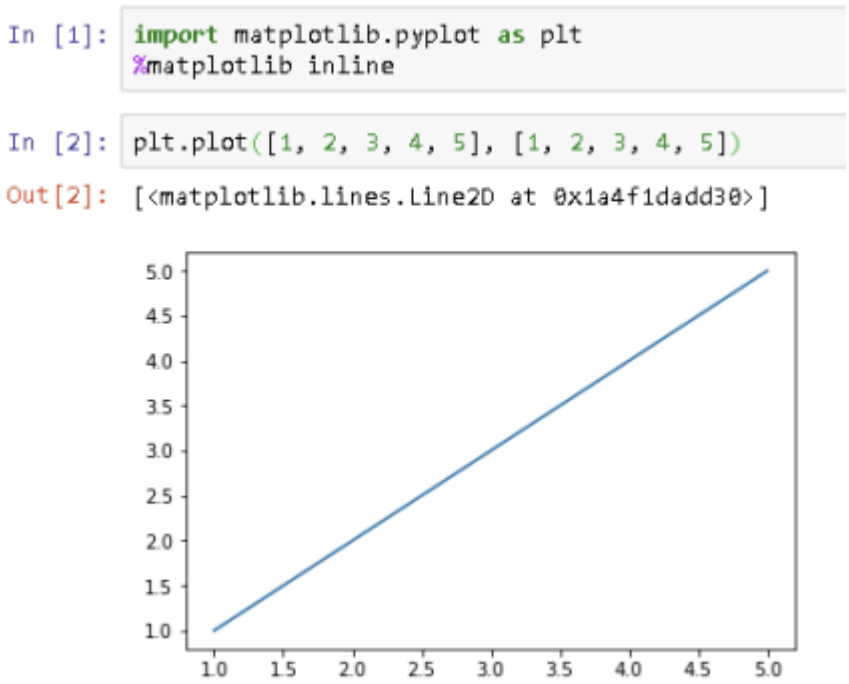

Рисунок 1 – Пример работы с Matplotlib в Anaconda

Если вы пишете код в .py файле, а потом запускаете его через вызов интерпретатора Python, то строка %matplotlib inline вам не нужна, используйте только импорт библиотеки. Пример, аналогичный тому, что представлен на рисунке выше, для отдельного Python-файла будет выглядеть так:

import matplotlib.pyplot as

plt plt.plot([1, 2, 3, 4, 5], [1, 2, 3, 4, 5])

plt.show()

В результате получите график в отдельном окне.

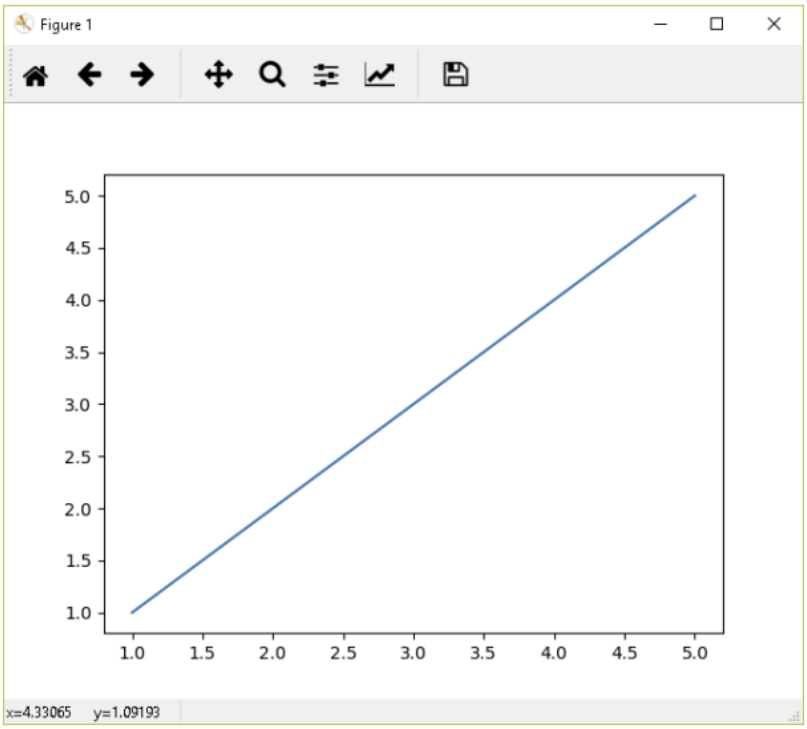

Рисунок 2 – График в отдельном окне

Далее мы не будем останавливаться на особенностях использования magic команды, просто запомните, что если вы используете Jupyter notebook при работе с Matplotlib вам обязательно нужно выполнить команду %matplotlib inline.

1. Изучить теоретический материал, представленный в теоретической части.
2. Установить Matplotlib.
3. Реализовать все примеры.
4. Постройте несколько линейных графиков на выбор.

Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.

2. Реализация каждого пункта подраздела «Методика и порядок выполнения работы» с приведением исходного кода программы, диаграмм и графиков для визуализации данных.

3. Ответы на контрольные вопросы.

4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Контрольные вопросы:

1. Визуализация данных и её использование. Достоинства и недостатки
2. Методы визуализации.
3. Цели визуализации данных?
4. Библиотека matplotlib для визуализации данных в Python.
5. Основные модули и классы matplotlib

ЛАБОРАТОРНАЯ РАБОТА №2. ПОСТРОЕНИЕ ГРАФИКОВ И ДИАГРАММ

Цель: научиться строить разные типы графиков, настраивать их внешний вид и освоиться в работе с этим инструментом.

Формируемые компетенции: ПК-2, ПК-6

Теоретическая часть

Далее мы будем использовать термин "график" для обозначения всего изображения, которое формирует Matplotlib (рисунок 1), и линии, построенной по заданному набору данных.

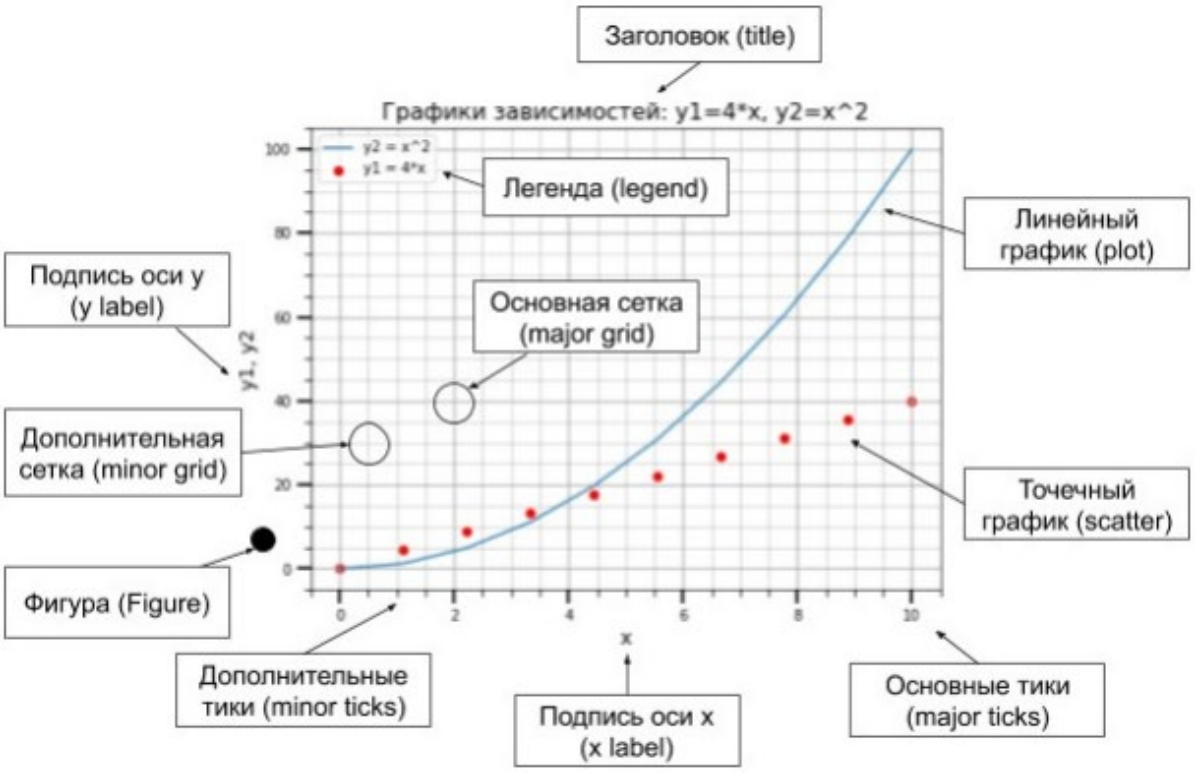

Рисунок 1 – Основные элементы графика

Корневым элементом, на котором Matplotlib строит изображение, является фигура (Figure). Всё, что перечислено на рисунке 1 — это элементы фигуры. Рассмотрим её составляющие более подробно.

На рисунке 1 представлены два графика — линейный и точечный. Matplotlib предоставляет огромное количество различных настроек, которые можно использовать для того, чтобы придать графику требуемый вид: цвет, толщина, тип, стиль линии и многое другое.

Вторым по важности элементом фигуры являются оси. Для каждой оси можно задать метку (подпись), основные (major) и дополнительные (minor) тики, их подписи, размер, толщину и диапазоны.

Сетка и легенда являются элементами фигуры, которые значительно повышают информативность графика. Сетка может быть основной (major) и дополнительной (minor). Каждому типу сетки можно задавать цвет, толщину линии и тип. Для отображения сетки и легенды используются соответствующие команды.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими программными средствами разработки (выбрать один или несколько програмных продуктов для практической реализации задач лабораторной работы): интерпретатор Python (Anaconda).

Методика и порядок выполнения работы

Ниже представлен код, с помощью которого был построен график, изображённый на рисунке 1:

```
import matplotlib.pyplot as plt
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
AutoMinorLocator)
import numpy as np

x = np.linspace(0, 10, 10)
y1 = 4*x
y2 = [i**2 for i in x]
fig, ax = plt.subplots(figsize=(8, 6))

ax.set_title('Графики зависимостей: y1=4*x, y2=x^2', fontsize=16)
ax.set_xlabel('x', fontsize=14)
ax.set_ylabel('y1, y2', fontsize=14)
ax.grid(which='major', linewidth=1.2)
ax.grid(which='minor', linestyle='--', color='gray', linewidth=0.5)
ax.scatter(x, y1, c='red', label='y1 = 4*x')
ax.plot(x, y2, label='y2 = x^2')
ax.legend()

ax.xaxis.set_minor_locator(AutoMinorLocator())
ax.yaxis.set_minor_locator(AutoMinorLocator())
ax.tick_params(which='major', length=10, width=2)
ax.tick_params(which='minor', length=5, width=1)

plt.show()
```

Практически все задачи, связанные с построением графиков, можно решить, используя возможности, которые предоставляет модуль pyplot. Для того, чтобы запустить любой из примеров, продемонстрированных в первой главе, вам предварительно нужно было импортировать pyplot из библиотеки Matplolib. В настоящее время среди пользователей принято импорт модуля pyplot производить следующим образом:

```
import matplotlib.pyplot as plt
```

Создатели Matplolib постарались сделать его похожим в использовании на MATLAB, так что если вы знакомы с последним, то разобраться с Matplolib будет проще.

Основным элементом изображения, которое строит pyplot, является фигура (Figure), на неё накладывается одно или более поле с графиками, оси координат, текстовые надписи и т.д. Для построения графика используется функция plot(). В самом минимальном варианте её можно использовать без параметров

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot()
```

В результате будет выведено пустое поле (см. рисунок 1)

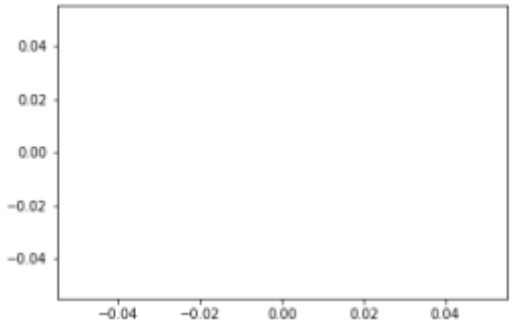

Рисунок 2 – Пустое поле

Далее команда импорта и magic-команда для Jupyter (первая и вторая строки, приведённой выше программы) приводиться не будут. Если в качестве параметра функции plot() передать список, то значения из этого списка будут отложены по оси ординат (ось y), а по оси абсцисс (ось x) будут отложены индексы элементов массива:

```
plt.plot([1, 7, 3, 5, 11, 1])
```

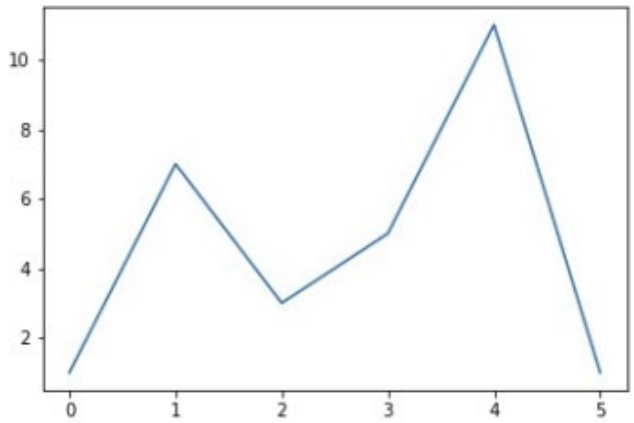

Рисунок 3 – Линейный график, построенный по значениям для оси Y

Для того чтобы задать значения по осям X и Y, необходимо в plot() передать два списка:

```
plt.plot([1, 5, 10, 15, 20], [1, 7, 3, 5, 11])
```

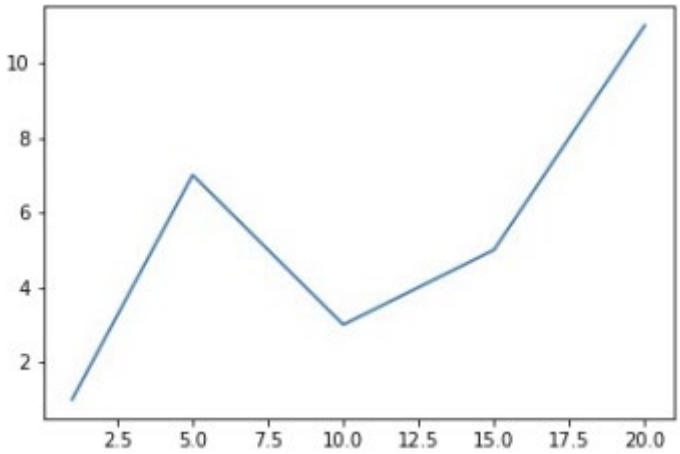

Рисунок 4 – Линейный график, построенный по значениям для оси Y

Построим несколько графиков на одном поле, для этого добавим квадратичную зависимость:

```
# Линейная зависимость
x = np.linspace(0, 10, 50)
y1 = x

# Квадратичная зависимость
y2 = [i**2 for i in x]
# Построение графика
plt.title('Зависимости: y1 = x, y2 = x^2') # заголовок
plt.xlabel('x')          # ось абсцисс
plt.ylabel('y1, y2')     # ось ординат
plt.grid()               # включение отображения сетки
plt.plot(x, y1, x, y2)   # построение графика
```

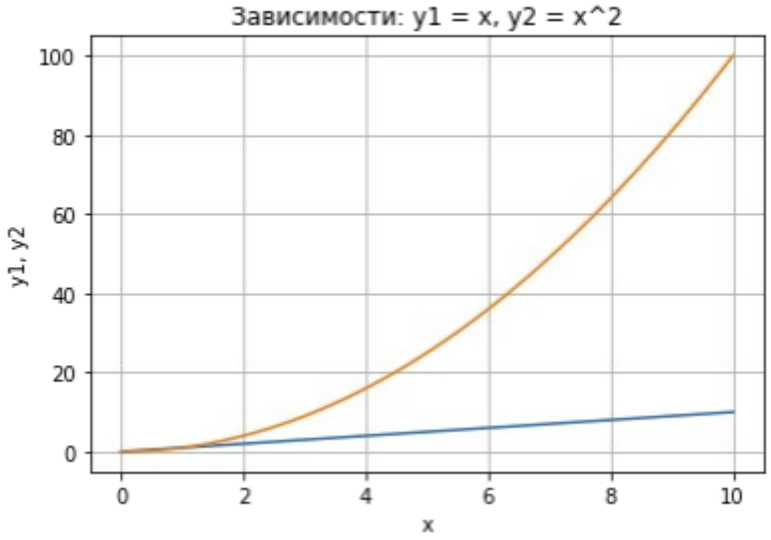

Рисунок 3 – Построение нескольких графиков на одном поле

В приведённом примере в функцию plot() последовательно передаются два массива для построения первого графика и два для построения второго, при этом, как вы можете заметить, для обоих графиков массив значений независимой переменной x один и то же.

Представление графиков на разных полях

Третья, довольно часто встречающаяся задача — это отображение двух или более различных полей, на которых может быть представлено несколько графиков.

Построим уже известные нам зависимости на разных полях:

```
x = np.linspace(0, 10, 50)
y1 = x                                  # Линейная зависимость
y2 = [i**2 for i in x]                  # Квадратичная зависимость
# Построение графиков
plt.figure(figsize=(9, 9))
plt.subplot(2, 1, 1)
plt.plot(x, y1)                         # построение графика
plt.title('Зависимости: y1 = x, y2 = x^2') # заголовок
plt.ylabel('y1', fontsize=14)           # ось ординат
plt.grid(True)                          # включение отображение сетки

plt.subplot(2, 1, 2)
plt.plot(x, y2)                         # построение графика
plt.xlabel('x', fontsize=14)            # ось абсцисс
plt.ylabel('y2', fontsize=14)           # ось ординат
plt.grid(True)                          # включение отображение сетки
```

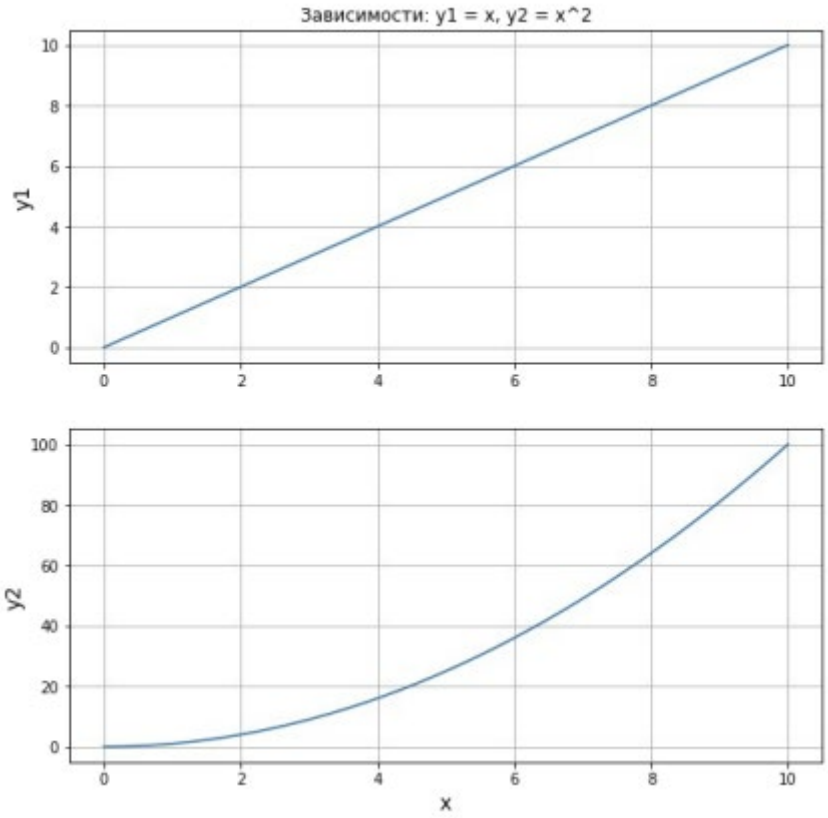

Рисунок 4 – Разделенные поля с графиками

Здесь мы воспользовались двумя новыми функциями:

• figure() — функция для задания глобальных параметров отображения графиков. В неё через параметр figsize передаётся кортеж из двух элементов, определяющий общий размер фигуры.

• subplot() — функция для задания местоположения поля с графиком. Существует несколько способов задания областей для вывода графиков. В примере мы воспользовались вариантом, который предполагает передачу трёх аргументов: первый аргумент — количество строк, второй — столбцов в формируемом поле, третий — индекс (номер поля, считаем сверху вниз, слева направо).

Остальные функции вам знакомы, дополнительно мы использовали параметр fontsize функций xlabel() и ylabel() для задания размера шрифта.

1. Изучить теоретический материал, представленный в теоретической части.
2. Повторить все примеры.
3. Постройте линейный график

Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.

2. Реализация каждого пункта подраздела «Методика и порядок выполнения работы» с приведением исходного кода программы, диаграмм и графиков для визуализации данных.

3. Ответы на контрольные вопросы.

4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Контрольные вопросы:

1. Для чего используются графики?
2. Линейный график - это?
3. Основные элементы графиков и диаграмм.
4. Редактирование и форматирование графиков и диаграмм?

ЛАБОРАТОРНАЯ РАБОТА №3. НАСТРОЙКА ЭЛЕМЕНТОВ ГРАФИКА.

Цель: научиться настраивать элементы графика.

Формируемые компетенции: ПК-2, ПК-6

Теоретическая часть

Наиболее часто используемые текстовые надписи на графике это:

• наименования осей;

• наименование самого графика;

• текстовые примечания на поле с графиком;

• легенда. Далее представлен обзор, перечисленных выше элементов.

Для задания подписи оси x используется функция xlabel(), оси y — ylabel(). Разберёмся с аргументами данных функций. Основными параметрами функций xlabel() и ylabel() являются:

xlabel (или ylabel): str

Текст подписи.

labelpad: численное значение либо None; значение по умолчанию: None

Расстояние между областью графика, включающим оси, и текстом подписи.

Функции xlabel() и ylabel() дополнительно принимают в качестве аргументов параметры конструктора класса matplotlib.text.Text (далее Text), вот некоторые из них:

fontsize или size: число либо значение из списка: {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}

Размер шрифта.

fontstyle: значение из списка: {'normal', 'italic', 'oblique'}

Стиль шрифта.

fontweight: число в диапазоне от 0 до 1000 либо значение из списка: {'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'}

Толщина шрифта.

color: один из доступных способов задания цвета.

Цвет текста подписи.

Пример использования:

```
plt.xlabel('Day', fontsize=15, color='blue')
```

Заголовок графика

Для задания заголовка графика используется функция title():

```
plt.title('Chart price', fontsize=17)
```

Из её параметров отметим следующие:

label: str

Текст заголовка.

loc: значение из набора: {'center', 'left', 'right'}

Выравнивание заголовка.

Для функции title() также доступны параметры конструктора класса Text, некоторые из них приведены в описании аргументов функций xlabel() и ylabel().

Текстовое примечание

За размещение текста на поле графика отвечает функция text(). Первый и второй её аргументы — это координаты позиции, третий — текст надписи, пример использования:

```
plt.text(1, 1, 'type: Steel')
```

Для более тонкой настройки внешнего вида текстового примечания используйте параметры конструктора класса Text.

Легенда

Легенда будет размещена на графике, если вызвать функцию legend().

Разместим на уже знакомом нам графике рассмотренные выше текстовые элементы:

```
x = [1, 5, 10, 15, 20]
y = [1, 7, 3, 5, 11]
plt.plot(x, y, label='steel price')
plt.title('Chart price', fontsize=15)
plt.xlabel('Day', fontsize=12, color='blue')
plt.ylabel('Price', fontsize=12, color='blue')
plt.legend()
plt.grid(True)
plt.text(15, 4, 'grow up!')
```

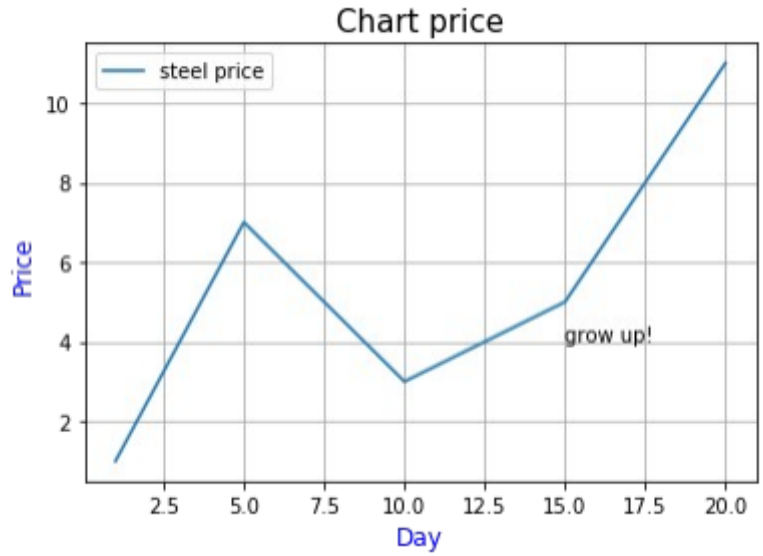

Рисунок 1 – Текстовые надписи на графики

К перечисленным опциям мы добавили сетку, которая включается с помощью функции grid(True).

Методика и порядок выполнения работы

В лабораторной работе будут рассмотрены основные параметры, которые можно использовать для изменения внешнего вида линейного графика.

Линейный график строится с помощью функции plot(), его внешний вид можно настроить непосредственно через аргументы указанной функции, например:

```
plt.plot(x, y, color='red')
```

Либо можно воспользоваться функцией setp():

```
plt.setp(color='red', linewidth=1)
```

Стиль линии графика задаётся через параметр linestyle, который может принимать значения из таблицы 1

Значение параметра	Описание
'-' или 'solid'	Непрерывная линия.
'--' или 'dashed'	Штриховая линия.
'-.' или 'dashdot'	Штрихпунктирная линия.
':' или 'dotted'	Пунктирная линия.
'None' или ' ' или ''	Не отображать линию.

Стиль линии можно передать сразу после списков с координатами без указания, что это параметр linestyle:

```
x = [1, 5, 10, 15, 20]
y = [1, 7, 3, 5, 11]
plt.plot(x, y, '--')
```

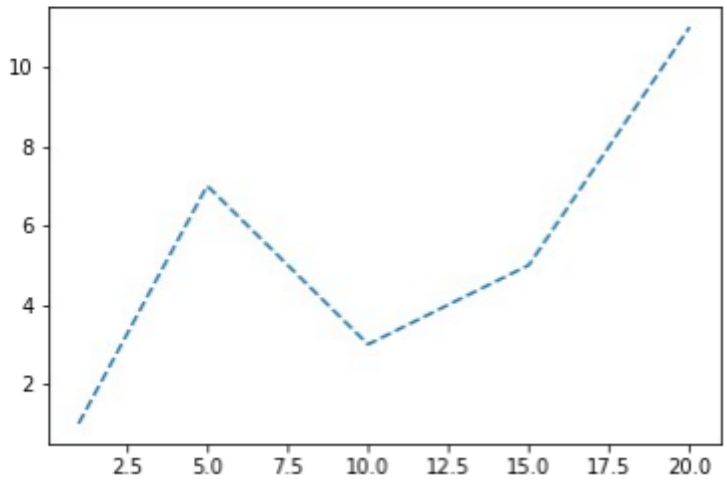

Рисунок 2 – Линия с примененным стилем

Другой вариант — это воспользоваться функцией setp():

```
x = [1, 5, 10, 15, 20]
y = [1, 7, 3, 5, 11]
line = plt.plot(x, y)
plt.setp(line, linestyle='--')
```

Результат будет тот же, что на рисунке 1.

Для того, чтобы вывести несколько графиков на одном поле, необходимо передать соответствующие наборы значений в функцию plot(). Построим несколько наборов данных и выведем их, задав различные стили линиям:

```
x = [1, 5, 10, 15, 20]
y1 = [1, 7, 3, 5, 11]
y2 = [i*1.2 + 1 for i in y1]
y3 = [i*1.2 + 1 for i in y2]
y4 = [i*1.2 + 1 for i in y3]
plt.plot(x, y1, '-', x, y2, '--', x, y3, '-.', x, y4, ':')
```

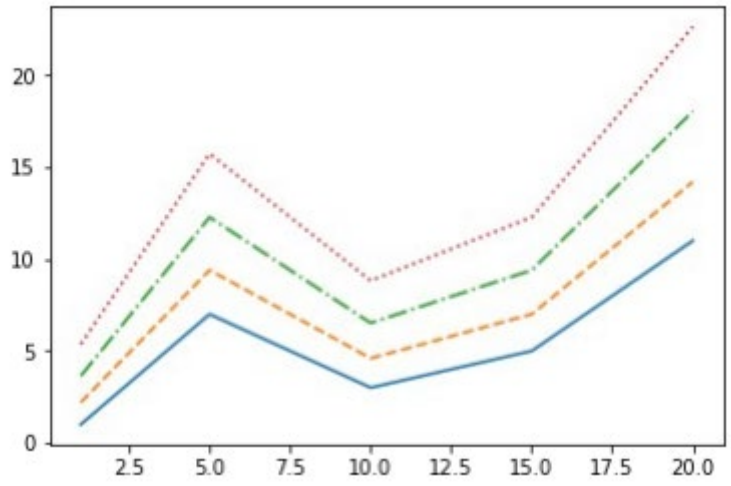

Рисунок 2 – Несколько графиков, построенных одной функцией plot()

Тот же результат можно получить, вызвав plot() отдельно для каждого набора данных. Если вы хотите представить графики изолированно друг от друга (каждый на своём поле), то используйте для этого функцию subplot()

```
plt.plot(x, y1, '-')
plt.plot(x, y2, '--')
plt.plot(x, y3, '-.')
plt.plot(x, y4, ':')
```

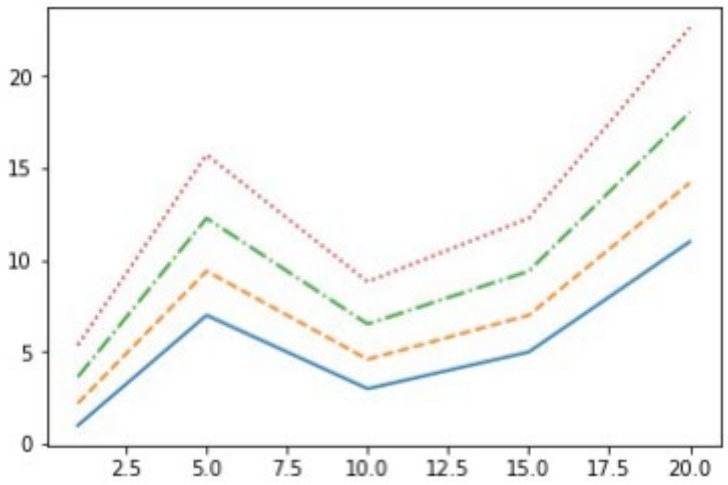

Рисунок 3 – Несколько графиков, построенных разными функциями plot()

Цвет линии графика задаётся через параметр color (или c, если использовать сокращённый вариант). Значение может быть представлено в одном из следующих форматов:

RGB или RGBA: кортеж значений с плавающей точкой в диапазоне [0, 1] (пример: (0.1, 0.2, 0.3);

RGB или RGBA: значение в hex формате (пример: '#0a0a0a');

строковое представление числа с плавающей точкой в диапазоне [0, 1] (определяет цвет в шкале серого) (пример: '0.7');

символ из набора: {'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'};

имя цвета из палитры X11/CSS4;

цвет из палитры xkcd (https://xkcd.com/color/rgb/), должен начинаться с префикса 'xkcd:';

цвет из набора Tableau Color (палитра T10), должен начинаться с префикса 'tab:'.

Если цвет задаётся с помощью символа из набора {'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'}, то он может быть совмещён со стилем линии в рамках параметра fmt функции plot(). Например: штриховая красная линия будет задаваться так: '--r', а штрихпунктирная зелёная так '-.g':

```
x = [1, 5, 10, 15, 20]
y = [1, 7, 3, 5, 11]
plt.plot(x, y, '--r')
```

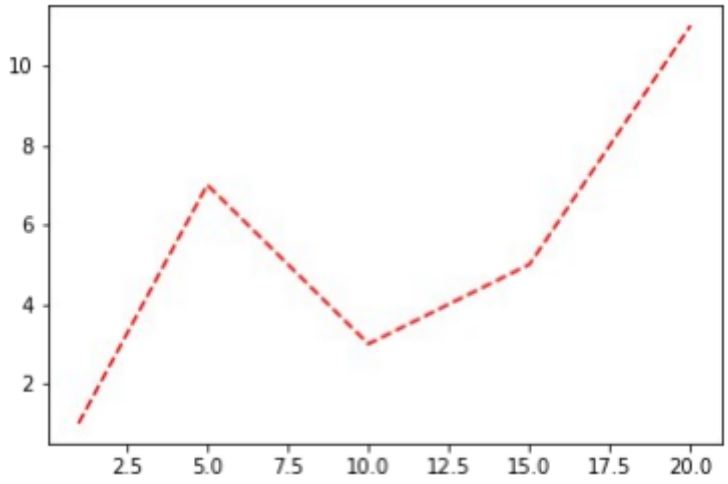

Рисунок 4 – График, представленный в виде штриховой красной линии

1. Изучить теоретический материал, представленный в теоретической части.
2. Повторить все примеры.
3. Постройте график. Измените дизайн кривой так, чтобы это была зелёная штриховая линия толщиной 1.2 pt. с красными ромбовидными маркерами и тонкой синей окантовкой.

Содержание отчета и его форма

1.Отчет по лабораторной работе должен содержать:

2.Номер и название лабораторной работы; задачи лабораторной работы.

3.Реализация каждого пункта подраздела «Методика и порядок выполнения работы» с приведением исходного кода программы, диаграмм и графиков для визуализации данных.

4.Ответы на контрольные вопросы.

5.Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Контрольные вопросы:

1. Каким образом подписывают оси графика? Приведите пример.
2. Как задать заголовок графика?
3. Как изменить начертание графика в виде пунктирной линии?
4. Приведите пример изменения цвета текста подписи графика?

ЛАБОРАТОРНАЯ РАБОТА №4. ВИЗУАЛИЗАЦИЯ ДАННЫХ. ОСНОВНЫЕ ТИПЫ ГРАФИКОВ И РАБОТА С НИМИ.

Цель: рассмотреть основные параметры, которые можно использовать для изменения внешнего вида линейного графика.

Формируемые компетенции: ПК-2, ПК-6

Теоретическая часть

До этого момента мы работали только с линейными графиками, функция plot() позволяет задать тип графика: линейный либо точечный. Для точечного графика дополнительно можно задать маркер, обозначающий положение точек.

Приведём пару примеров:

```
plt.plot(x, y, 'ro')
```

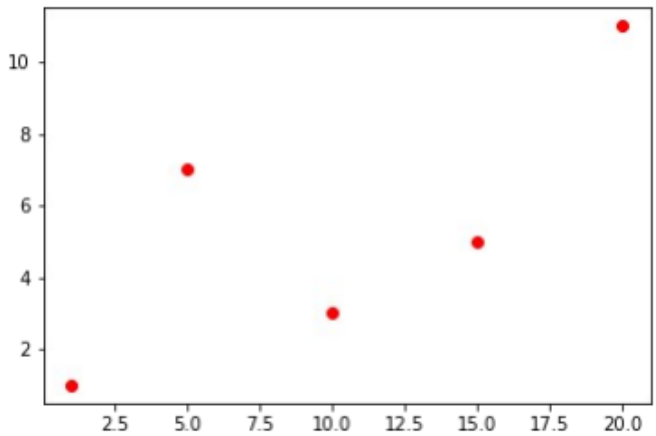

Рисунок 1 – График, состоящий из круглых красных точек

```
plt.plot(x, y, 'bx')
```

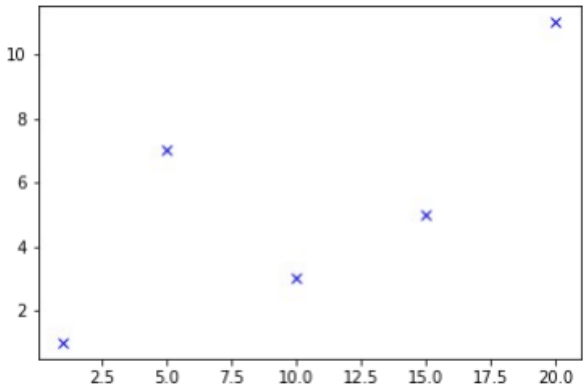

Рисунок 2 – График, состоящий из синих x-символов

Существуют три основных подхода к размещению графиков на разных полях:

• использование функции subplot() для указания места размещения поля с графиком;

• использование функции subplots() для предварительного задания сетки, в которую будут укладываться поля;

• использование GridSpec, для более гибкого задания геометрии размещения полей с графиками на сетке.

Самый простой способ вывести графики на отдельных полях — это использовать функцию subplot() для задания мест их размещения. До этого момента мы не работали с Фигурой (Figure) напрямую, значения ее параметров, задаваемые по умолчанию, нас устраивали. Для решения текущей задачи придётся один из параметров — размер подложки, задать вручную. За это отвечает аргумент figsize функции figure(), которому присваивается кортеж из двух float-элементов, определяющих высоту и ширину подложки.

После задания размера указывается местоположение: куда будет установлено поле с графиком с помощью функции subplot().

Доступны следующие варианты вызова subplot():

subplot(nrows, ncols, index)

• nrows: int ◦ Количество строк.

• ncols: int

◦ Количество столбцов.

• index: int

◦ Местоположение элемента. subplot(pos)

• pos: int

◦ Позиция. Задаётся в виде трехзначного числа, содержащего информацию о количестве строк, столбцов и индексе, например, число 212 означает: подготовить разметку с двумя строками и одним столбцом, элемент вывести в первую позицию второй строки.

Второй вариант можно использовать, если количество строк и столбцов сетки не более 10, в ином случае лучше обратиться к первому варианту.

Рассмотрим на примере работу с данными функциями:

```
# Исходный набор данных
x = [1, 5, 10, 15, 20]
y1 = [1, 7, 3, 5, 11]
y2 = [i*1.2 + 1 for i in y1]
y3 = [i*1.2 + 1 for i in y2]
y4 = [i*1.2 + 1 for i in y3]
# Настройка размеров подложки
plt.figure(figsize=(12, 7))
# Вывод графиков
plt.subplot(2, 2, 1)
plt.plot(x, y1, '-')

plt.subplot(2, 2, 2)
plt.plot(x, y2, '--')
plt.subplot(2, 2, 3)
plt.plot(x, y3, '-.')
plt.subplot(2, 2, 4)
plt.plot(x, y4, ':')
```

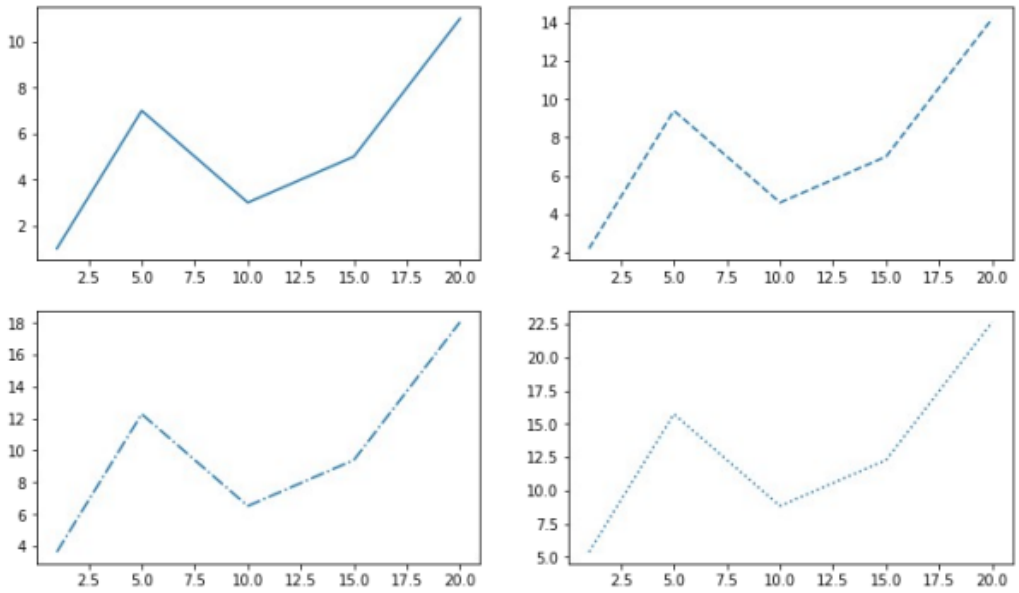

Рисунок 3 – Размещение графиков на отдельных полях

Решим эту же задачу, используя второй вариант вызова subplot():

```
plt.subplot(221)
plt.plot(x, y1, '-')
plt.subplot(222)
plt.plot(x, y2, '--')
plt.subplot(223)
plt.plot(x, y3, '-.')
plt.subplot(224)
plt.plot(x, y4, ':')
```

В результате будет получен график, аналогичный приведённому на рисунке 3.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими программными средствами разработки (выбрать один или несколько програмных продуктов для практической реализации задач лабораторной работы): интерпретатор Python (Anaconda).

Методика и порядок выполнения работы

Работа с функцией subplots()

Неудобство использования последовательного вызова функций subplot() заключается в том, что каждый раз приходится указывать количество строк и столбцов сетки. Для того, чтобы этого избежать, можно воспользоваться функцией subplots(), из всех ее параметров нас интересуют только первые два, через них передаётся количество строк и столбцов сетки. Функция subplots() возвращает два объекта, первый — это Figure, подложка, на которой будут размещены поля с графиками, второй — объект (или массив объектов) Axes, через который можно получить полных доступ к настройке внешнего вида отображаемых элементов.

Решим задачу вывода четырёх графиков с помощью subplots():

```
fig, axs = plt.subplots(2, 2, figsize=(12, 7))
axs[0, 0].plot(x, y1, '-')
axs[0, 1].plot(x, y2, '--')
axs[1, 0].plot(x, y3, '-.')
axs[1, 1].plot(x, y4, ':')
```

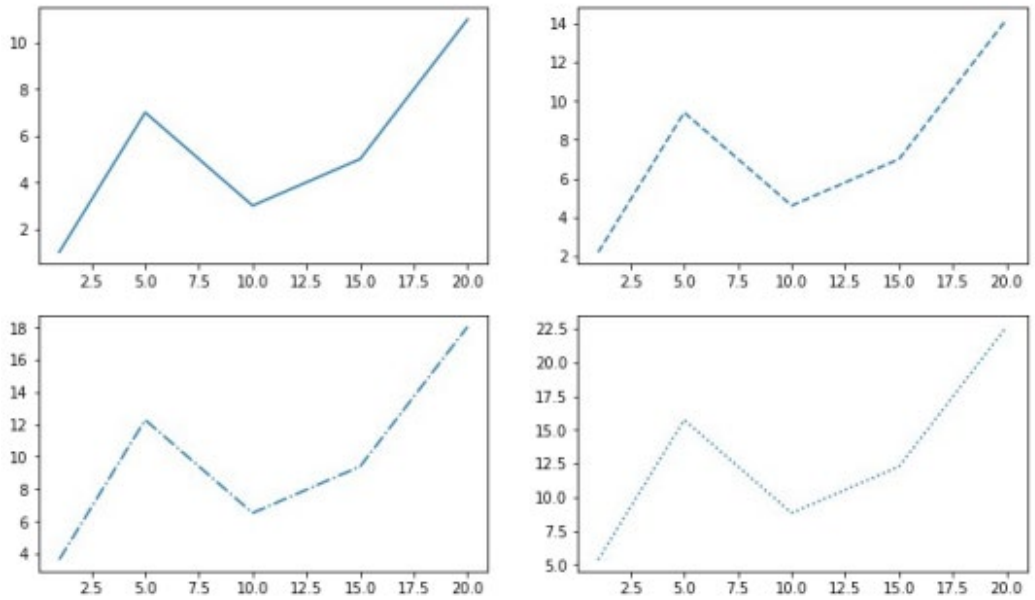

Рисунок 4 – Размещение графиков на отдельных полях

Настройка элементов графика

В данном разделе будут рассмотрены следующие темы: отображение легенды, настройка её расположения на графике, настройка внешнего вида легенды.

Для отображения легенды на графике используется функция legend(). Возможны следующие варианты её вызова: legend() legend(labels) legend(handles, labels) В первом варианте в качестве меток для легенды будут использоваться метки, указанные в функциях построения графиков (параметр label):

```
x = [1, 5, 10, 15, 20]
y1 = [1, 7, 3, 5, 11]
y2 = [4, 3, 1, 8, 12]
plt.plot(x, y1, 'o-r', label='line 1')
plt.plot(x, y2, 'o-.g', label='line 1')
plt.legend()
```

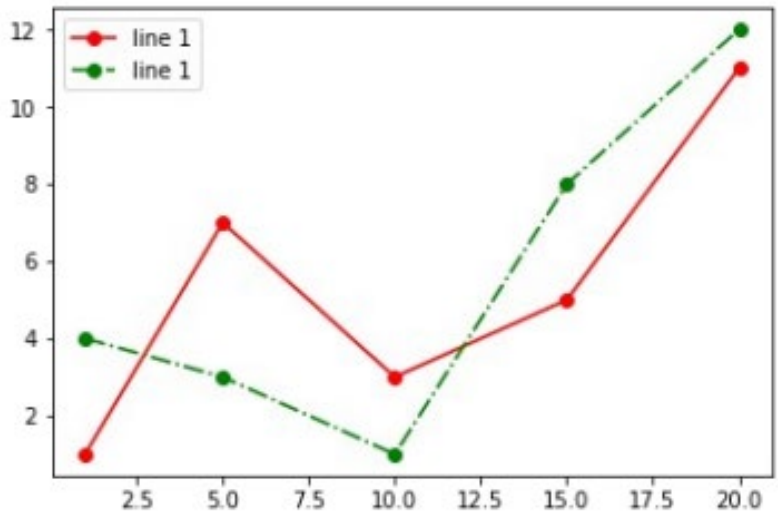

Рисунок 5 – Легенда на графике (пример 1)

Второй вариант позволяет самостоятельно указать текстовую метку для отображаемых данных:

```
plt.plot(x, y1, 'o-r')
plt.plot(x, y2, 'o-.g')
plt.legend(['L1', 'L2'])
```

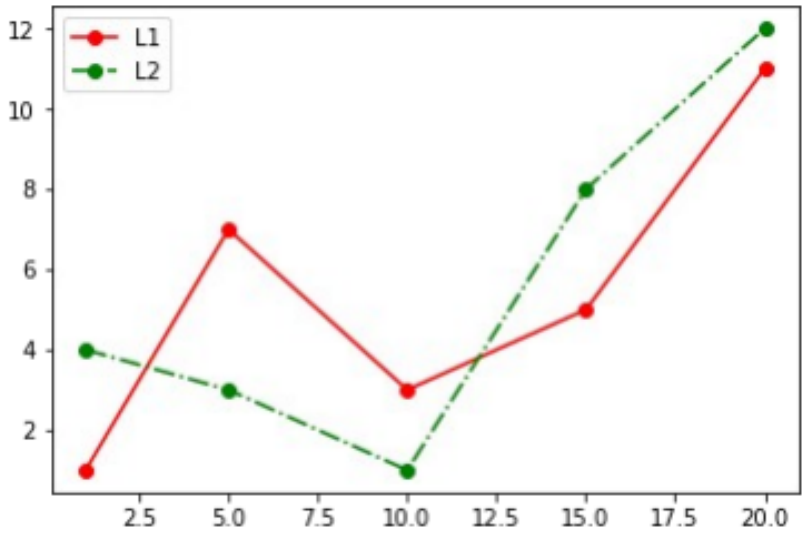

Рисунок 5 – Легенда на графике (пример 2)

В третьем варианте можно вручную указать соответствие линий и текстовых меток:

```
line1, = plt.plot(x, y1, 'o-b')
line2, = plt.plot(x, y2, 'o-.m')
plt.legend((line2, line1), ['L2', 'L1'])
```

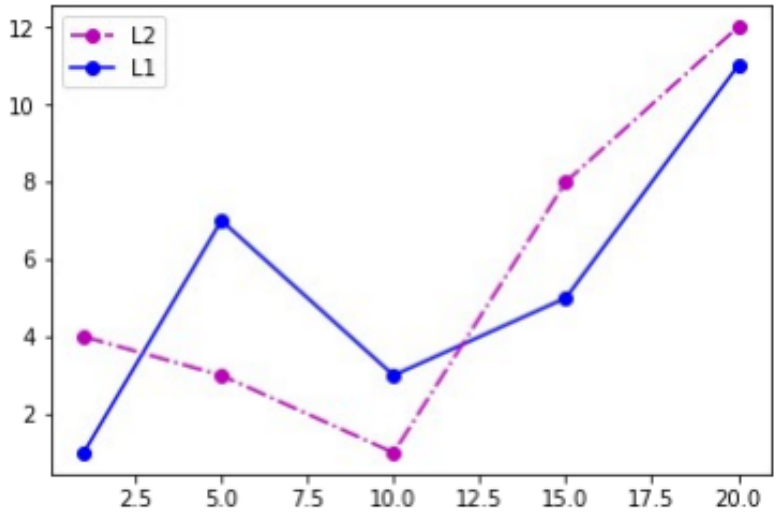

Рисунок 6 – Легенда на графике (пример 3)

Место расположения легенды определяется параметром loc, который может принимать одно из значений, указанных в таблице1.

Строковое описание	Код
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

Ниже представлен пример, демонстрирующий различные варианты расположения легенды через параметр loc:

```
locs = ['best', 'upper right', 'upper left', 'lower left',
        'lower right', 'right', 'center left', 'center right',
        'lower center', 'upper center', 'center']
plt.figure(figsize=(12, 12))
for i in range(3):
    for j in range(4):
        if i*4+j < 11:
            plt.subplot(3, 4, i*4+j+1)
            plt.title(locs[i*4+j])
            plt.plot(x, y1, 'o-r', label='line 1')


    plt.plot(x, y2, 'o-.g', label='line 2')
    plt.legend(loc=locs[i*4+j])
else:
    break
```

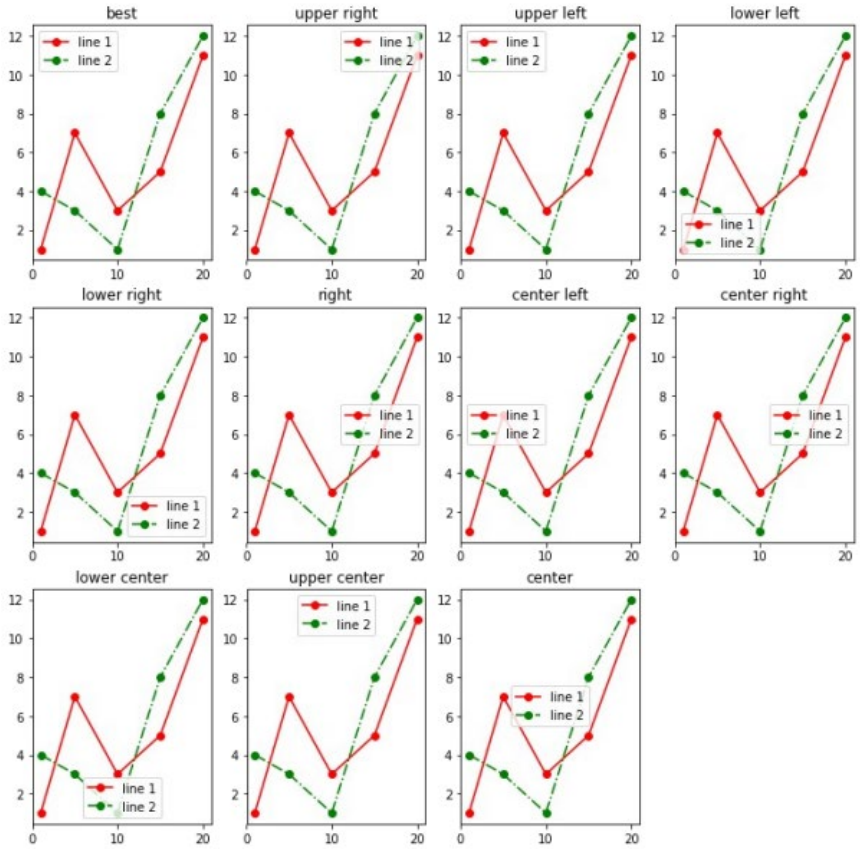

Рисунок 7 – Различные варианты расположения легенды на графике

Для более гибкого управления расположением легенды можно воспользоваться параметром bbox_to_anchor функции legend().

Этому параметру присваивается кортеж, состоящий из четырёх или двух элементов:

bbox_to_anchor = (x, y, width, height)

bbox_to_anchor = (x, y) где x, y — это координаты расположения легенды;

width — ширина;

height — высота.

Пример использования параметра bbox_to_anchor:

```
plt.plot(x, y1, 'o-r', label='line 1')
plt.plot(x, y2, 'o-.g', label='line 1')
plt.legend(bbox_to_anchor=(1, 0.6))
```

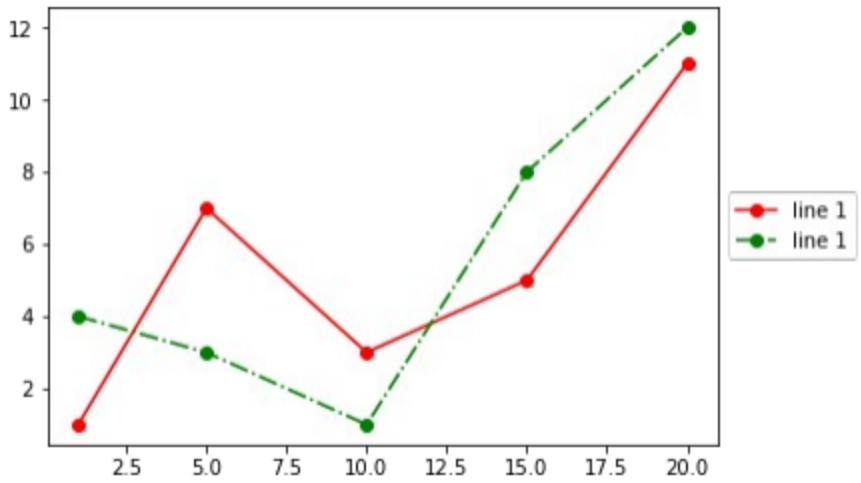

Рисунок 8 – Расположение легенды вне поля графика

1. Изучить теоретический материал, представленный в теоретической части.
2. Повторить все примеры из теоретической части.
3. Используя pyplot.legend, измените размер шрифта Legend
4. Создайте легенду с цветным полем.
5. Создайте график и удалите границу легенды на нем.

Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.

2. Реализация каждого пункта подраздела «Методика и порядок выполнения работы» с приведением исходного кода программы, диаграмм и графиков для визуализации данных.

3. Ответы на контрольные вопросы.

4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Контрольные вопросы:

1. Для чего нужна легенда на графике?
2. Как изменить цвет шрифта легенды?
3. Как разместить легенду вне графика?
4. Основные элементы графиков и диаграмм.
5. Редактирование и форматирование графиков и диаграмм.

ЛАБОРАТОРНАЯ РАБОТА №5. ВИЗУАЛИЗАЦИЯ ДАННЫХ. ОСНОВНЫЕ ТИПЫ ДИАГРАММ. СТОЛБЧАТЫЕ ДИАГРАММЫ. КРУГОВЫЕ ДИАГРАММЫ

Цель: научиться строить разные типы диаграмм, настраивать их внешний вид и освоиться в работе с этим инструментом.

Формируемые компетенции: ПК-2, ПК-6

Теоретическая часть

Перед выполнением лабораторной работы необходимо ознакомиться с теорией построения круговых и столбчатых диаграмм, используя следующие источники: [1-3]. Особое внимание необходимо уделить репозитарию [1] с исходными кодами.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими программными средствами разработки (выбрать один или несколько програмных продуктов для практической реализации задач лабораторной работы): интерпретатор Python (Anaconda).

Методика и порядок выполнения работы

Для визуализации категориальных данных хорошо подходят столбчатые диаграммы. Для их построения используются функции:

• bar() — вертикальная столбчатая диаграмма;

• barh() — горизонтальная столбчатая диаграмма

Построим простую диаграмму

```
np.random.seed(123)
groups = [f'P{i}' for i in range(7)]
counts = np.random.randint(3, 10, len(groups))
plt.bar(groups, counts)
```

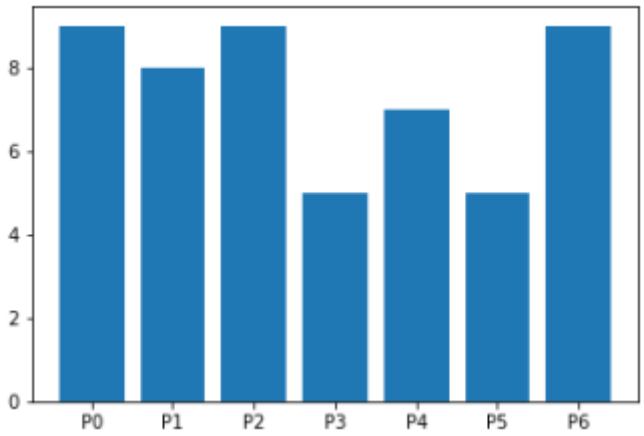

Рисунок 1 – Вертикальная столбчатая диаграмма

Если заменим bar() на barh(), то получим горизонтальную диаграмму: plt.barh(groups, counts)

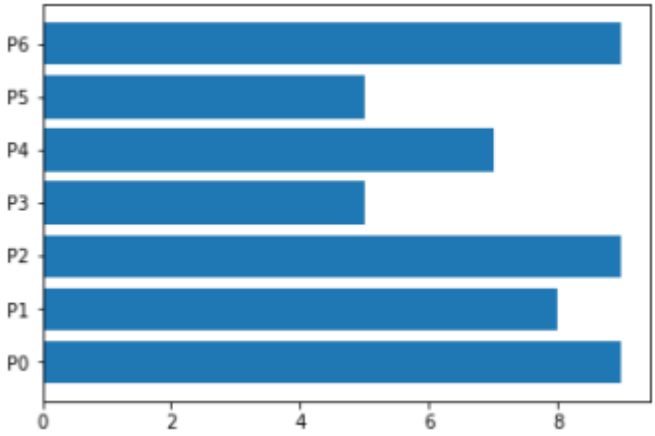

Рисунок 2 – Горизонтальная столбчатая диаграмма

Рассмотрим более подробно параметры функции bar():

Основные параметры:

• x: массив

◦ x-координаты столбцов.

• height: скалярная величина или массив

◦ Высоты столбцов.

• width: скалярная величина, массив или optional

◦ Ширина столбцов.

• bottom: скалярная величина, массив или optional

◦ y-координата базы.

• align: {'center', 'edge'}, optional; значение по умолчанию: 'center' ◦ Выравнивание по координате x.

Дополнительные параметры:

• color: цвет 9 , набор цветовых элементов или optional

◦ Цвет столбцов диаграммы.

• edgecolor: цвет10, набор цветовых элементов или optional

◦ Цвет границы столбцов.

• linewidth: скалярная величина, массив или optional

◦ Ширина границы.

• tick_label: str, массив или optional

◦ Метки для столбца.

• xerr, yerr: скалярная величина, массив размера shape(N,), shape(2, N) или optional

◦ Величина ошибки для графика. Выставленное значение прибавляется/удаляется к верхней (правой — для горизонтального графика) границе. Может принимать следующие значения:

▪ скаляр: симметрично +/- для всех баров;

▪ shape(N,): симметрично +/- для каждого бара;

▪ shape(2, N): выборочного - и + для каждого бара. Первая строка содержит нижние значения ошибок, вторая строка — верхние;

▪ None: не отображать значения ошибок. Это значение используется по умолчанию.

• ecolor: цвет10, набор цветовых элементов или optional; значение по умолчанию: 'black' ◦ Цвет линии ошибки.

• log: bool, optional; значение по умолчанию: False ◦ Включение логарифмического масштаба для оси y.

• orientation: {'vertical', 'horizontal'}, optional ◦ Ориентация: вертикальная или горизонтальная.

Пример, демонстрирующий работу с параметрами bar():

```
import matplotlib.colors as mcolors
bc = mcolors.BASE_COLORS

np.random.seed(123)
groups = [f'P{i}' for i in range(7)]
counts = np.random.randint(0, len(bc), len(groups))
width = counts*0.1
colors = [['r', 'b', 'g'][int(np.random.randint(0, 3, 1))] for _ in
counts]

plt.bar(groups, counts, width=width, alpha=0.6, bottom=2, color=colors,
edgecolor='k', linewidth=2)
```

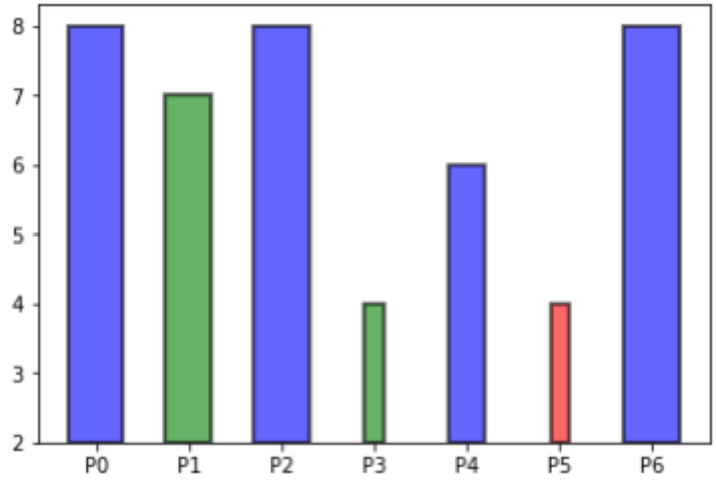

Рисунок 3 – Модифицированная столбчатая диаграмма

Групповые столбчатые диаграммы

Используя определенным образом подготовленные данные, можно строить групповые диаграммы:

```
cat_par = [f'P{i}' for i in range(5)]
g1 = [10, 21, 34, 12, 27]
g2 = [17, 15, 25, 21, 26]
width = 0.3
x = np.arange(len(cat_par))
```

```
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, g1, width, label='g1')
rects2 = ax.bar(x + width/2, g2, width, label='g2')
ax.set_title('Пример групповой диаграммы')
ax.set_xticks(x)
ax.set_xticklabels(cat_par)
ax.legend()
```

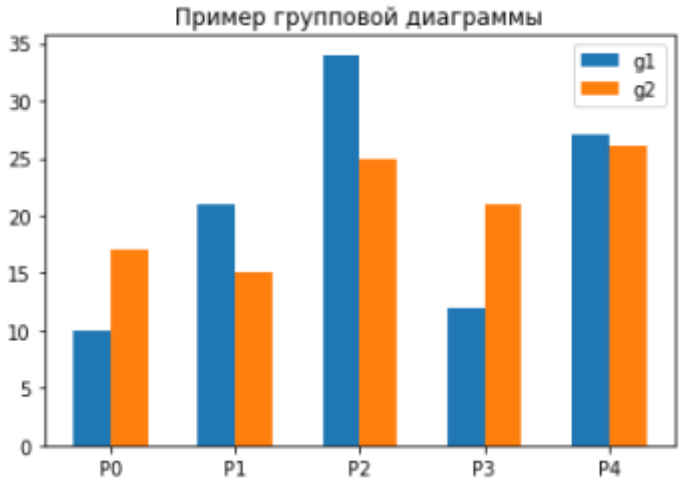

Рисунок 4 – Групповая столбчатая диаграмма

Круговые диаграммы

Круговые диаграммы — это наглядный способ показать доли компонентов в наборе. Они идеально подходят для отчётов, презентаций и т.п. Для построения круговых диаграмм в Matplotlib используется функция pie(). Пример диаграммы:

```
vals = [24, 17, 53, 21, 35]
labels = ['Ford', 'Toyota', 'BMW', 'AUDI', 'Jaguar']
fig, ax = plt.subplots()
ax.pie(vals, labels=labels)
ax.axis('equal')
```

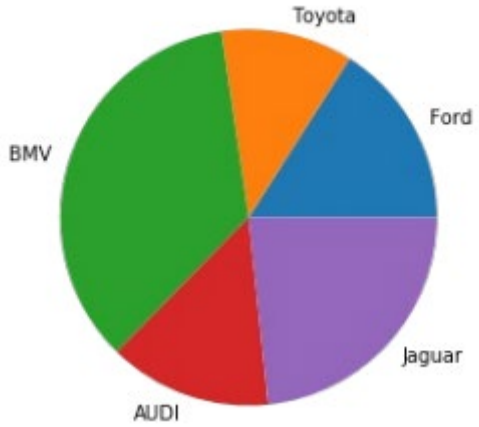

Рисунок 5 – Круговая диаграмма

Рассмотрим параметры функции pie():

• x: массив ◦ Массив с размерами долей.

• explode: массив, optional; значение по умолчанию: None

◦ Если параметр не равен None, то часть долей, которые перечислены в передаваемом значении, будут вынесены из диаграммы на заданное расстояние. Пример такой диаграммы:

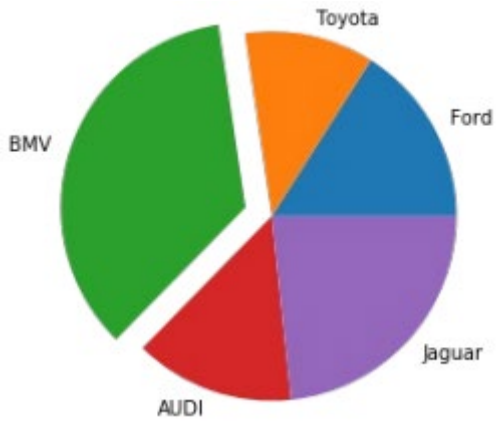

Рисунок 6 – Круговая диаграмма с отделенным сектором

labels: list, optional; значение по умолчанию: None

◦ Текстовые метки долей.

colors: массив цветовых элементов11 , optional; значение по умолчанию: None

Цвета долей.

autopct: str, функция, optional; значение по умолчанию: None

Формат текстовой метки внутри доли, текст — это численное значение показателя, связанного с конкретной долей.

pctdistance: float, optional; значение по умолчанию: 0.6

Расстояние между центром каждой доли и началом текстовой метки, которая определяется параметром autopct.

shadow: bool, optional, значение по умолчанию: False

Отображение тени для диаграммы.

labeldistance: float, None, optional; значение по умолчанию: 1.1

Расстояние, на котором будут отображены текстовые метки долей. Если параметр равен None, то метки не будут отображены.

startangle: float, optional; значение по умолчанию: None

Угол, на который нужно повернуть диаграмму против часовой стрелки относительно оси x.

radius: float, optional; значение по умолчанию: None

Величина радиуса диаграммы.

counterclock: bool, optional; значение по умолчанию: True

Направление вращения: по часовой или против часовой стрелки.

wedgeprops: dict, optional; значение по умолчанию: None

Словарь параметров, определяющих внешний вид долей (см. класс matplotlib.patches.Wedge).

textprops: dict, optional; значение по умолчанию: None

Словарь параметров, определяющих внешний вид текстовых меток (см. класс matplotlib.text.Text).

center: список значений float, optional; значение по умолчанию: (0, 0)

Центр диаграммы

frame: bool, optional; значение по умолчанию: False

Если параметр равен True, то вокруг диаграммы будет отображена рамка.

rotatelabels: bool, optional; значение по умолчанию: False

Если параметр равен True, то текстовые метки будут повернуты на заданный угол.

Пример, демонстрирующий работу с параметрами функции pie():

```
vals = [24, 17, 53, 21, 35]
labels = ['Ford', 'Toyota', 'BMW', 'AUDI', 'Jaguar']
explode = (0.1, 0, 0.15, 0, 0)
fig, ax = plt.subplots()
ax.pie(vals, labels=labels, autopct='%1.1f%%', shadow=True,
explode=explode, wedgeprops={'lw':1, 'ls':'--','edgecolor':'k'},
rotatelabels=True)
ax.axis('equal')
```

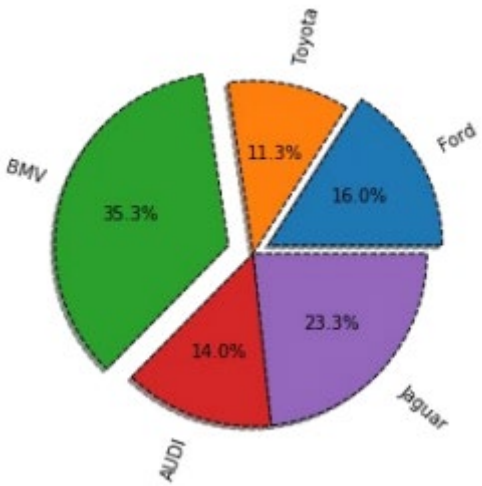

Рисунок 7 – Модифицированная круговая диаграмма

1. Изучить теоретический материал, представленный в теоретической части.
2. Повторить все примеры из теоретической части.
3. Построить круговую диаграмму с распределением кафе по различным городам России (не менее 10 городов). Статистику взять из открытых источников в сети интернет.

Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.

2. Реализация каждого пункта подраздела «Методика и порядок выполнения работы» с приведением исходного кода программы, диаграмм и графиков для визуализации данных.

3. Ответы на контрольные вопросы.

4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Контрольные вопросы:

1. Как построить круговую диаграмму с отдельными секторами?
2. Что такое ступенчатая диаграмма?
3. Как настроить цвет столбцов столбчатой диаграммы?
4. Что значит вложенная круговая диаграмма
5. Как построить круговую диаграмму с отверстием в центре?

ЛАБОРАТОРНАЯ РАБОТА №6. ПОСТРОЕНИЕ 3D-ГРАФИКОВ.

Цель: познакомиться с этапами построения 3D-графиков.

Формируемые компетенции: ПК-2, ПК-6

Теоретическая часть

До этого момента все графики, которые мы строили, были двумерные, а Matplotlib позволяет строить также 3D-графики. Импортируем необходимые модули для работы с 3D: import matplotlib.pyplot as plt from mpl_toolkits.mplot3d import Axes3D

Рассмотрим некоторые из инструментов для построения 3D-графиков.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими программными средствами разработки (выбрать один или несколько програмных продуктов для практической реализации задач лабораторной работы): интерпретатор Python (Anaconda).

Методика и порядок выполнения работы

Для построения линейного графика используется функция plot() из Axes3D:

```
Axes3D.plot(self, xs, ys, *args, zdir='z', **kwargs)
```

Параметры функции *Axes3D.plot*:

- `xs`, `ys`: *1D*-массивы
  - Координаты точек по осям *x* и *y*.
- `zs`: число или *1D*-массив
  - *z* координаты. Если передано скалярное значение, то оно будет присвоено всем точкам графика.
- `zdir`: `{'x', 'y', 'z'}`; значение по умолчанию: `'z'`
  - Ось, которая будет принята за *z* направление.
- `**kwargs`
  - Дополнительные аргументы, аналогичные тем, что используются в функции `plot()` для построения двумерных графиков.

```
x = np.linspace(-np.pi, np.pi, 50)
y = x
z = np.cos(x)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z, label='parametric curve')
```

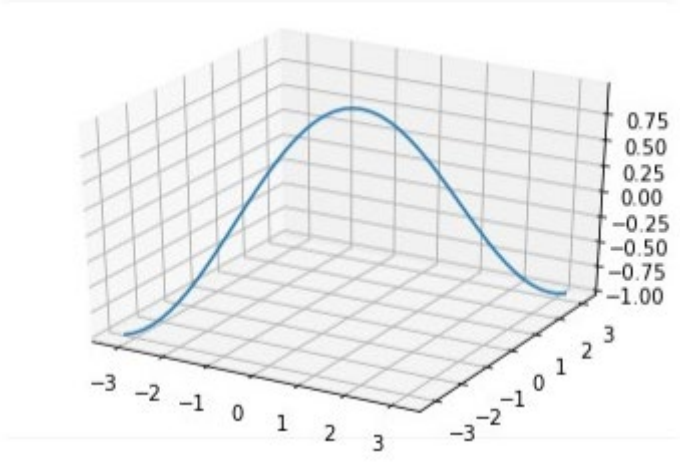

Рисунок 1 – Демонстрация работы функции Axes3D.plot()

Для построения диаграммы рассеяния используется функция scatter() из Axes3D:

```
Axes3D.scatter(self, xs, ys, zs=0, zdir='z', s=20, c=None,
depthshade=True, *args, **kwargs)
```

Параметры функции Axes3D.scatter():

• xs, ys: 1D-массив ◦ Координаты точек по осям x и y.

• zs: число или 1D-массив, optional; значение по умолчанию: 0

◦ Координаты точек по оси z. Если передано скалярное значение, то оно будет присвоено всем точкам графика.

zdir: {'x', 'y', 'z', '-x', '-y', '-z'}, optional; значение по умолчанию 'z'

Ось, которая будет принята за z направление.

• s: число или массив, optional; значение по умолчанию 20

◦ Размер маркера.

• c: цвет14, массив чисел, массив цветовых элементов, optional

◦ Цвет маркера. Возможные значения:

▪ строковое значение цвета для всех маркеров;

▪ массив строковых значений цвета;

▪ массив чисел, которые могут быть отображены в цвета через функции cmap и norm;

▪ 2D-массив, элементами которого являются RGB или RGBA;

• depthshade: bool, optional ◦ Затенение маркеров для придания эффекта глубины.

• **kwargs

◦ Дополнительные аргументы, аналогичные тем, что используются в функции scatter() для построения двумерных графиков.

```
np.random.seed(123)
x = np.random.randint(-5, 5, 40)
y = np.random.randint(0, 10, 40)
z = np.random.randint(-5, 5, 40)
s = np.random.randint(10, 100, 20)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, s=s)
```

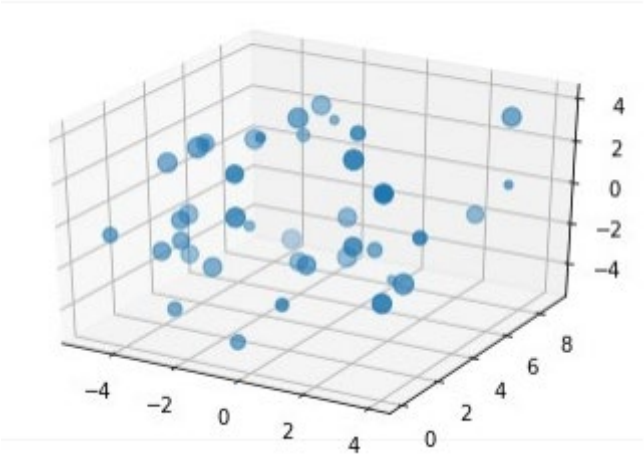

Рисунок 2 – Демонстрация работы функции Axes3D.scatter()

1. Изучить теоретический материал, представленный в теоретической части.
2. Повторить все примеры из теоретической части.
3. Федеральная служба государственной статистики приводит данные об уровне денежных доходов населения в целом по России и по субъектам Российской Федерациии (руб./месяц) 1.

Используя подготовленный csv-файл, создайте изображение с 2-мя диаграммами:

круговая диаграмма: средний доход по федеральным округам РФ;

столбчатая диаграмма: доход по субъектам РФ в рамках федерального округа (номер округа - ЦИФРА_СБ % 9 + 1).

Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.

2. Реализация каждого пункта подраздела «Методика и порядок выполнения работы» с приведением исходного кода программы, диаграмм и графиков для визуализации данных.

3. Ответы на контрольные вопросы.

4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Контрольные вопросы:

1. Основные типы диаграмм: столбчатая (линейчатая) диаграмма.
2. Основные типы диаграмм: гистограмма.
3. Основные правила выбора диаграммы и дополнительные возможности matplotlib.
4. Как построить 3d диаграмму.

ЛАБОРАТОРНАЯ РАБОТА №7. ПОСТРОЕНИЕ 3D-ГРАФИКОВ. ВИДЫ ПОВЕРХНОСТЕЙ

Цель: познакомиться с разновидностями поверхностей
Формируемые компетенции: ПК-2, ПК-6

Теоретическая часть

Цветокоррекция – внесение изменений в цвет оригинала. Многие профессионалы относят к цветокоррекции те процедуры, которые не связаны с изменением сюжета изображения.

В более узком смысле цветокоррекция – это такое преобразование изображения, объекта или фрагмента, когда новый цвет обрабатываемого пикселя зависит от старого значения этого пикселя и не зависит от соседних пикселей.

DS 36-3 C, что в отличии от инструмента Hue/Saturation даст гарантированное попадание в цвет при печати или попадание в цвет дизайна, где планируется размещение данного изображения.

Для построения каркасной поверхности используется функция plot_wireframe():

plot_wireframe(self, X, Y, Z, *args, **kwargs)

Параметры функции Axes3D.wireframe():

• X, Y, Z: 2D массивы

◦ Данные для построения поверхности.

• rcount, ccount: int, значение по умолчанию: 50

◦ Максимальное количество элементов каркаса, которое будет использовано в каждом из направлении.

• rstride, cstride: int ◦ Параметры, определяющие величину шага, с которым будут браться элементы строки / столбца из переданных массивов.

Параметры rstride, cstride и rcount, ccount являются взаимоисключающими.

• **kwargs

◦ Дополнительные аргументы, определяемые Line3DCollection.

```
u, v = np.mgrid[0:2*np.pi:20j, 0:np.pi:10j]
x = np.cos(u)*np.sin(v)
y = np.sin(u)*np.sin(v)
z = np.cos(v)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(x, y, z)
ax.legend()
```

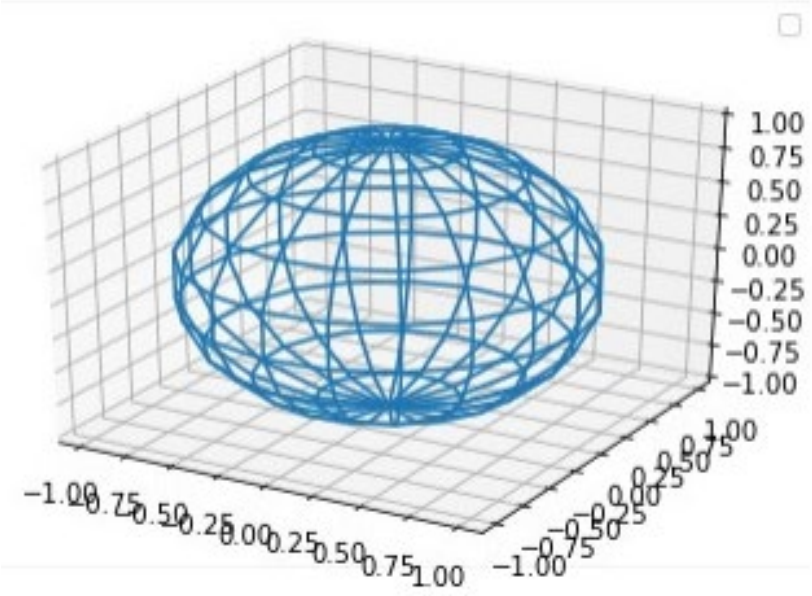

Рисунок 1 – Каркасная поверхность

Для построения поверхности используйте функцию plot_surface(): plot_surface(self, X, Y, Z, *args, norm=None, vmin=None, vmax=None, lightsource=None, **kwargs)

Параметры функции Axes3D.plot_surface():

• X, Y, Z: 2D массивы

◦ Данные для построения поверхности.

• rcount, ccount: int

◦ см. rcount, ccount в “5.3 Каркасная поверхность”.

• rstride, cstride : int ◦ см.rstride, cstride в “5.3 Каркасная поверхность”.

• color: color

◦ Цвет для элементов поверхности.

• cmap: Colormap

◦ Colormap для элементов поверхности.

• facecolors: массив элементов color

◦ Индивидуальный цвет для каждого элемента поверхности.

• norm: Normalize

◦ Нормализация для colormap.

• vmin, vmax: float

◦ Границы нормализации.

• shade: bool; значение по умолчанию: True

◦ Использование тени для facecolors.

• lightsource: LightSource

◦ Объект класса LightSource - определяет источник света, используется, только если shade = True.

• **kwargs ◦ Дополнительные аргументы, определяемые Poly3DCollection.

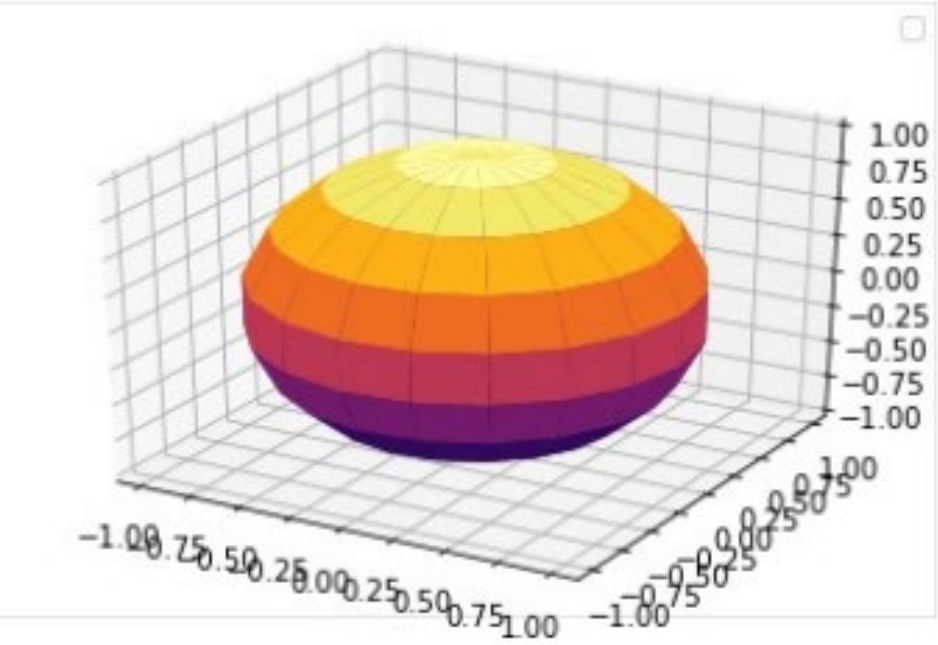

Рисунок 2 – Каркасная поверхность

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими программными средствами разработки (выбрать один или несколько програмных продуктов для практической реализации задач лабораторной работы): интерпретатор Python (Anaconda).

Методика и порядок выполнения работы

1. Изучить теоретический материал, представленный в теоретической части.
2. Повторить все примеры из теоретической части.
3. Построить диаграмму с каркасной поверхностью на выбор.

Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.

2. Реализация каждого пункта подраздела «Методика и порядок выполнения работы» с приведением исходного кода программы, диаграмм и графиков для визуализации данных.

3. Ответы на контрольные вопросы.

4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Контрольные вопросы

Что такое каркасная поверхность?

Что такое поверхностный график?

СПИСОК ОСНОВНОЙ ЛИТЕРАТУРЫ

1. Абдрахманов М.И..Python.Визуализация данных: Matplotlib. Seaborn. Mayavi/ Абдрахманов М.И.– 2020. – 413 с.

2. Уэс, Маккинли. Python и анализ данных Электронный ресурс / Маккинли Уэс ; пер. А. А. Слинкин. - Python и анализ данных,2021-04-19. - Саратов : Профобразование, 2017. - 482 c. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-4488-0046-7, экземпляров неограниченно.

3. Сирота, А.А. Методы и алгоритмы анализа данных и их моделирование в MATLAB / А.А. Сирота. - СПб.: BHV, 2016. - 384с.

СПИСОК ДОПОЛНИТЕЛЬНОЙ ЛИТЕРАТУРЫ

4. Мэтиз Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения / Мэтиз Э. – СПб.: Питер, 2017. – 496 с.

5. Железны Д. Говори на языке диаграмм: пособие по визуальным коммуникациям / Джин Железны; пер. с англ. [А.Мучника и Ю.Корнилович] – 5-е изд. – М.: Манн, Иванов и Фербер, 2012. – 304с.

6. Макшанов, А.В. Технологии интеллектуального анализа данных: Учебное пособие / А.В. Макшанов, А.Е. Журавлев. - СПб.: Лань, 2018. - 212 с

7. Смирнова Е.А., Смирнов М.А. Введение в цифровую культуру: учебное пособие / Смирнова Е.А., Смирнов М.А. – Череповец. 2021. – 201 с.

8. http://matplotlib.org. - Библиотека matplotlib.

9.http://rus-linux.net/MyLDP/BOOKS/Architecture-Open-Source-Applications/Vol-2/matplotlib-02.html - Обзор архитектуры библиотеки matplotlib.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕУЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

по организации самостоятельной работы обучающихся
по дисциплине **«Инструментальные средства визуализации данных»**
Направление подготовки 09.03.02 «Информационные системы и технологии»
Направленность (профиль) «Прикладное программирование в интеллектуальных информационных системах»

Ставрополь
2026

Введение

Подготовка бакалавра – сложный, комплексный процесс, в котором органически сочетается решение всех основных задач – образовательных, воспитательных и развивающих. Деятельность преподавателя в учебном процессе – ведущая. Однако, как бы ни были совершенны применяемые преподавателем в учебном процессе форм и методов обучения, без активной познавательной деятельности самих обучаемых они не могут давать желаемых результатов. Поэтому особое значение приобретает проблема сознательного отношения самих студентов к приобретению знаний, умений и навыков в результате самостоятельной работы.

Самостоятельная работа - это познавательная деятельность, выполняемая студентами самостоятельно, при частичном руководстве преподавателя. Самостоятельная работа предусмотрена учебной программой. Самостоятельная работа является управляемым процессом. Благодаря этому познавательно-практическая деятельность студентов становится целенаправленной, творчески осмысленной, содержательной.

Процесс управления самостоятельной работы служит главным целям обучения:

- усвоение, закрепление, совершенствование знаний в объеме вузовских программ;
- приобретение умений и навыков, составляющих содержание подготовки выпускника высшей школы;
- максимальная активность каждого обучающегося в овладении профессиональных компетенций (ПК-2, 6), формируемых в результате освоения дисциплины.

1. Самостоятельная работа студентов на лабораторных и практических занятиях

1.1. Аудиторные занятия

Задачи:

- • расширение, углубление и детализация знаний, полученных на лекциях;
- • повышение уровня усвоения учебного материала (от уровня информации,
- полученной на лекциях, до уровня знаний за счет привития умений и навыков);
- • развитие научного мышления и речи студентов;
- • проверка и учет знаний;
- • развитие познавательной активности и привитие навыков самостоятельной
- работы, особенно с дополнительной и специальной литературой;
- • привитие навыков ведения коллективной беседы, участия в творческой
- дискуссии, умения аргументированно отстаивать свои взгляды.

Основное требование к подготовке и участию студентов в занятиях заключается в том, что студент должен самостоятельно готовиться к занятиям и творчески в них участвовать.

Этапы подготовки к занятиям включают:

- повторение уже имеющихся знаний по конспекту, а затем по учебнику;
- углубление знаний по теме с использованием рекомендованной
- литературы;
- выполнение конкретного задания (решение задач, составление отчетов и т.п.).

Особое место занимают лабораторные работы. Главная цель которых –

быть связующим звеном теории изучаемой дисциплины с практической реализацией, что позволяет углублять и закреплять теоретические положения, получаемые студентами на лекции, проверять их применение в практике экспериментальным путем, знакомить студентов с оборудованием, приборами, вычислительной техникой, изучать на практике методы научных исследований.

Студенты обеспечиваются методическими указаниями к лабораторным работам, содержащими теоретическую информацию и конкретные практические задания.

Для студентов младших курсов указания даются подробно и детализировано. Для студентов старших курсов предусматривается большая самостоятельность при выполнении заданий. Т.е. они должны уже демонстрировать переход от уровня умений к уровню навыков.

Оформление лабораторных работ должно быть максимально приближено к уровню, на котором ведется экспериментальная научно-исследовательская работа в конкретной предметной области.

1.2. Самостоятельная работа студентов во внеаудиторное время.

Кроме наиболее распространенных в вузах форм самостоятельной работы во внеаудиторное время (написание рефератов, контрольных работ, курсовых работ/проектов и т.д.) учебные планы включают такие виды и соответственно такой компонент фонда оценочных средств, как собеседование (по конкретным вопросам) и выполнение предлагаемых тем для самостоятельной учебно-исследовательской работы.

2. Указания студентам по подготовке к собеседованию и материалов по темам самостоятельной работы

2.1. Указания представлены в виде рекомендаций студентам в ходе изучения вопросов, выносимых на собеседование и материалов тем самостоятельной работы.

Своевременное и качественное выполнение самостоятельной работы базируется на соблюдении настоящих рекомендаций и изучении рекомендованной литературы. Студент может дополнить список использованной литературы современными источниками, не представленными в списке рекомендованной литературы, и в дальнейшем использовать собственные подготовленные учебные материалы при написании рефератов, курсовых и выпускных квалификационных работ.

2.2. Рекомендации

2.2.1. Прежде чем приступить к изучению вопросов для собеседования и предложенной темы самостоятельной работы, необходимо.

Во-первых:

- уяснить суть вопроса (темы) на самостоятельную работу;
- провести подбор рекомендованной литературы;
- составить план работы, в котором определяются основные пункты предстоящей подготовки.

Составление плана дисциплинирует и повышает организованность в работе.

Во-вторых:

- начинать надо с изучения рекомендованной литературы.

Необходимо помнить, что на лекции обычно рассматривается не весь материал, а только его часть. Остальная его часть восполняется в процессе самостоятельной работы. В связи с этим работа с рекомендованной литературой обязательна (см. Приложение 1). Особое внимание при этом необходимо обратить на содержание основных положений и выводов, объяснение явлений и фактов, уяснение практического приложения рассматриваемых теоретических вопросов.

В процессе этой работы студент должен стремиться понять и запомнить основные положения рассматриваемого материала, примеры, поясняющие его, а также разобраться в иллюстративном материале.

Заканчивать подготовку следует составлением плана (оглавления) по изучаемому материалу. Это позволит дать целостный, сжатый ответ при собеседовании и представить целостную картину по предложенной тематике.

В процессе самостоятельной работы рекомендуется взаимное обсуждение материала (между студентами), во время которого закрепляются знания, а также приобретается практика в изложении и разъяснении полученных знаний, развивается речь.

При необходимости студент может обращаться за консультацией к преподавателю. Идя на консультацию, студенту необходимо хорошо продумать вопросы, которые требуют разъяснения.

При подготовке к собеседованию и выполнению самостоятельной работы по теме студент должен вести записи. В результате у него создается свой индивидуальный фонд дополнительных материалов для быстрого повторения прочитанного, для накопления знаний. Особенно важны и полезны записи тогда, когда в них находят отражение мысли, возникшие при самостоятельной работе.

Основные формы записи: план (простой и развернутый), выписки, тезисы.

Результаты этих записей могут быть представлены в различных формах.

План – это схема прочитанного материала, краткий (или подробный)

перечень вопросов, отражающих структуру и последовательность материала:

краткий

- воспроизведение наиболее важных положений и фактов источника;

подробный (развернутый)

- детализированный план, в котором достаточно подробные записи приводятся по тем пунктам плана, которые нуждаются в пояснении;
- это четко и кратко изложенные (сформулированные) основные положения в результате глубокого осмысливания материала;
- в нем могут присутствовать выписки, цитаты, тезисы.

Подробно составленный план позволяет эффективно использовать материал и получить желаемый результат.

2.2.2. При отчете студента о проделанной самостоятельной работе по предложенной теме, преподаватель может предложить студенту алгоритм действий, рекомендовать еще раз внимательно прочитать весь материал и все записи по теме самостоятельной работы, тщательно продумать свое устное выступление.

2.2.3. Устный отчет должен строиться свободно, убедительно и аргументированно.

2.2.4. Преподаватель следит, чтобы отчет не сводился к простому воспроизведению текста, не допускается и простое чтение. Необходимо, чтобы студент проявлял собственное отношение к тому, о чем он говорит, высказывал свое личное мнение, понимание, обосновывал его и мог сделать правильные выводы из сказанного. При этом студент может обращаться к записям собранного материала, использовать знания факты и наблюдения современной жизни и т. д.

2.2.5. В заключение преподаватель оценивает результаты работы студента по данной теме.

3. Критерий оценки самостоятельной работы студента

Оценка «зачтено» при ответах на вопросы при собеседовании и по итогам устной защиты выполненной самостоятельной работы выставляется студенту, если студент показал знание учебного материала. При этом возможно допустил неточные формулировки основных понятий и терминов или ряд незначительных неточностей, не исказивших существенно суть ответа.

Оценка «не зачтено» при ответах на вопросы при собеседовании и по итогам устной защиты выполненной самостоятельной работ выставляется студенту, если студент не знает значительной части учебного материала, допускает неправильную формулировку основных понятий и терминов, существенно исказившие суть вопроса.

Оценка «не зачтено» выставляется также, если студент не отвечает на вопросы при собеседовании, отсутствует план выполненной работы, нет объяснений основных понятий и терминов.

4. Заключение

Успешное освоение курса предполагает активное, творческое участие студента путем планомерной, повседневной работы как на лекциях, лабораторных (практических) занятиях, так и к подготовке ответов на вопросы к собеседованию и выполнению самостоятельной работы по предлагаемым темам.

Приложение 1

Рекомендации студентам по изучению рекомендованной литературы

Изучение дисциплины следует начинать с проработки рабочей программы по изучаемой дисциплине, особое внимание, уделяя целям и задачам, структуре и содержанию курса.

Студентам рекомендуется получить учебную литературу по дисциплине, необходимую для эффективной работы на всех видах аудиторных занятий, а также для самостоятельной работы по изучению дисциплины.

Приложение 2

Примерная тематика заданий для самостоятельной работы студентов

Студенты выполняют самостоятельную работу в соответствии с индивидуальным заданием, которое определяется примерными темами.

Примерные темы самостоятельной работы

1. Визуализация данных и её использование. Достоинства и недостатки.
2. Методы визуализации.
3. Качество визуализации.
4. Базовые правила и принципы визуализации.
5. Технические средства визуализации (мониторы, экраны, проекторы).
6. Графики. Диаграммы. Гистограмма.
7. Диаграмма времени (шкала времени). Диаграмма визуализации процесса (блок-схема).
8. Матрицы. Карты и картограммы.
9. Инфографика. Презентации. Дашборды.
10. Виды поддерживаемых диаграмм и графиков.
11. Визуализация 3D-данных.
12. Использование инструментов ParaView и VisIt.
13. Визуализация данных в физике, биологии, медицине
14. Использование библиотек Plotly и Bokeh для создания интерактивных графиков.
15. Построение интерактивных дашбордов с использованием Dash (Python) или Shiny (R).
16. Визуализация данных в реальном времени.
17. Построение линейных графиков для временных рядов.
18. Использование анимаций для визуализации изменений во времени.
19. Визуализация сезонности и трендов.
20. Роль визуализации данных в анализе и принятии решений.

21. Основные принципы эффективной визуализации.
22. Типы данных и подходящие для них виды визуализаций.